Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Žan Pajek Arambašić

Implementacija modela informacijskega sistema za podporo poslovanju študentskih klubov

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM
PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: prof. dr. Marko Bajec

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V okviru diplomskega dela razvijte informacijski sistem za potrebe študentskega kluba. Na konkretnem primeru zajemite zahteve ter jih analizirajte. Rezultat analize naj bo model informacijskega sistema, ki bo kar se da splošen in uporaben tudi za druge tovrstne organizacije. Na tej osnovi razvijte načrt ter implementirajte informacijski sistem. Na koncu komentirajte izbrano metodo dela in identificirajte možne izboljšave.

Kazalo

Slike

Povzetek

Naslov: Implementacija modela informacijskega sistema za podporo poslovanju študentskih klubov

Avtor: Žan Pajek Arambašić

Informacijski sistemi so vseprisotni v današnjem življenju. Dobro zasnovan informacijski sistem lahko uporabnikom olajša delo in jim ponudi prave informacije ob pravem času. Zato je ključno, da so ti sistemi skrbno načrtovani in kasneje izdelani po načrtu.
Glede na namen sistema, njegovo delovanje ter uporabo sistem uporablja različne tehnologije, da lahko deluje. Poleg tega izbrane tehnologije zaradi svojih specifik in omejitev določajo, kako bo model sistema implementiran v sami tehnologiji in obratno. Tu se pri izdelavi sistema pojavi vprašanje, kako dobro implementirati model glede na ciljno tehnologijo.
Namen naloge je razjasniti uporabo ter izdelavo sistema na podlagi modela. Z enostavnim primerom modela informacijskega sistema za podporo poslovanju študentskim klubom bomo predstavili, kako poteka proces modeliranja ter implementacije modela sistema.

Ključne besede: informacijski sistemi, modeliranje, implementacija modela.

Abstract

Title: Implementation of a information system model for student organisations

Author: Žan Pajek Arambašić

Information systems are all present in today's world. A well designed information system can help users by delivering the right information at the right time.
Modeling and later implementation of that model are therefore paramount for a good end product. In this work we will therefore present a model of an information system in the form of a modern web application. By comparing the model and the actual system we will show and explain the differences that occur, problems and potential solutions.
The goal is to find and solve the most common problems that programmers and architects face when designing and implementing an information system.

Keywords: information systems, modeling, implementation.

Poglavje 1

Uvod

Informacijski sistem je v širšem pomenu besede organiziran sistem za zbiranje, obdelavo in sporočanje informacij. Informacijski sistemi zavzemajo več oblik, s katerimi se srečujemo vsak dan. Mobilne in spletne aplikacije, programi, ki jih uporabljamo v službi, informacijske table, ki jih opazujemo, ko čakamo na avtobus ali vlak - vse to so informacijski sistemi različnih oblik.
Model informacijskega sistema opisuje delovanje, zgradbo in izgled informacjskega sistema. Tu večina programerjev najprej pomisli na razredni diagram specifikacije UML ali na ER diagram, ki opisuje obliko podatkov. Vendar je model veliko več. Je skupek shem, grafov, skic in tehničnih besedil, ki opisujejo vse pomembne aspekte delovanja informacijskega sistema. Dobro zasnovan model naj bi bil čim popolnejši in le malo stvari bi prepustil naključju ali lastni presoji. Proces modeliranja informacijskih sistemov je opisan v različnih specifikacijah. Najbolj znana, ki podrobno opisuje in določa modele informacijskega sistema je univerzalni modelirni jezik UML (angl. Unified Modeling Language). Podrobnost opisa, proces opisa ter proces izdelave modela pa so določeni v različnih metologijah dela. Najbolj znana metologija dela, ki opisuje tudi področje modeliranja je Rational Unified Process oz. RUP.
Naslednji logični korak po izdelanem modelu sistema je torej izdelava sistema oz. implementacija modela. Tu pa se pojavi izziv: kako čim bolje prenesti

model v realnost? Odgovor na to vprašanje je odvisen od tehnologij, ki so uporabljene, od metologije dela, ki narekuje način dela ter od posameznika ali skupine, ki model interpretira. V tem diplomskem delu bomo anaizirali implementacijo modela informacijskega sistema v obliki spletne aplikacije za podporo poslovanju študentskim klubom z imenom Kurnik. Kratko bomo opisali postopek izdelave modela spletne aplikacije ter vse dele modela tudi obrazložili. S primerjavo modela in implementacije bomo pojasnili razlike med njima, zakaj je do njih prišlo, ter poskusili odgovoriti na tipična vprašanja, ki nastanejo pri implementaciji informacijskega sistema.

1.1 Pomen modeliranja informacijskih sistemov

Modeliranje informacijskih sistemov je študija izdelave in uporabe formalnih opisov — modelov za opis ter popis delovanja informacijskih sistemov. Proces obsega modeliranje poslovnih procesov, komunikacijskih, računalniških ter drugih sistemov z namenom enotnega razumenvanja njihovega delovanja. V sodelovanju s standardnim načinom modeliranja ter uporabo modelirnega jezika lahko natančno opišemo delovanje sistema. Tako lahko vsak, ki razume specifikacijo preko modela sistema, pridobi podrobno razumevanje o delovanju sistema.

Če govorimo o informacijskih sistemih je modeliranje pomembno na nivoju načrtovanja, izdelave, uporabe ter vzdrževanja sistema. Natančen model sistema izključi dvoumnost, ki je lahko posledica uporabe naravnega jezika ter omogoči, da se vsi vpleteni s sistemom enoumno sporazumejo o sistemu.

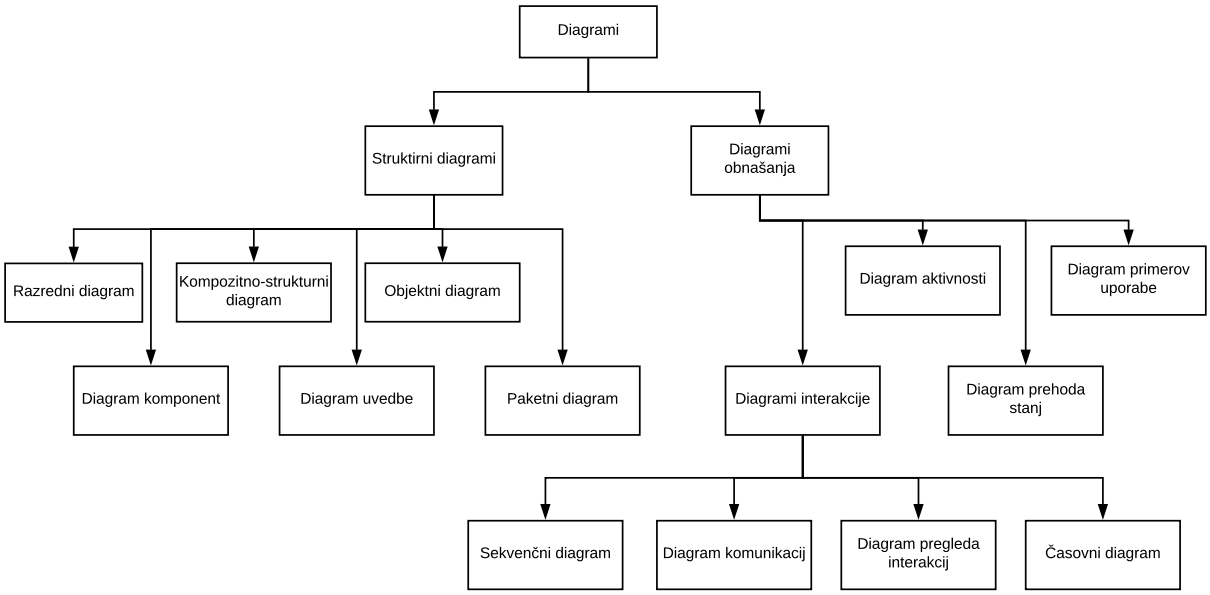
Za izdelavo kvalitetnega informacijskega sistema je model tako izredno pomemben, saj omogoči boljše razumevanje med izdelovalcem in naročnikom sistema.

1.2 Univerzalni modelirni jezik

Univerzalni modelirni jezik (angl. Unified Modeling Language) oz. UML je skupina grafičnih notacij, ki se uporablja za namene opisa, razvoja ter uporabe programske opreme še posebej, če ta temelji na principu objektnega programiranja. Omogoča standardno vizualizacijo različnih vidikov sistema. Specifikacija se je razvila iz težnje po standardizaciji opisovanja programske opreme. Prva verzija je izšla leta 1994, trenutno najnovejša verzija specifikacije je UML 2.5. Ta je izšla leta 2015 [8].

UML razvija ter upravlja skupina Object Management Group (OMG). To je skupina podjetji, ki je bila ustvarjena prav iz težnje po standardizaciji opisov med organizacijami ter podjetji.

UML opisuje več diagramskih tehnik ter procesov za opis mnogih vidikov sistemov. Te diagramske tehnike se delijo na 2 tipa: diagrame, ki pojasnjujejo strukturo sistema ter tiste, ki pojasnjujejo obnašanje sistema [8].



Slika 1.1: Hierarhija UML diagramov.

V našem primeru bomo Kurnik predstavili z pomočjo več diagramskih tehnik UML in sicer:

- Diagram primerov uporabe
- ER Diagram
- Komponentni diagrami

1.3 Drugi diagrami

Poleg diagramov, ki so izdelani po specifikaciji UML, lahko za potrebe izdelave sistema izdelamo tudi druge diagrame. Tu je potrebna dodatna previdnost, saj ostalih diagramov ne določa standard, zato jih je moč razumeti na več načinov. Cilj še vedno ostaja: čim bolj jasno ponazoriti različne aspekte sistema.
UML diagrami so zaradi obsežnega standarda včasih neprimerni za učinkovito komunikacijo, še posebej če se udeleženci ne spoznajo na UML standard. Enostavnejše diagramske tehnike lahko prav tako dobro prenesejo željen pomen, če so podkrepljene z razlago [13].

1.3.1 Model uporabniškega vmesnika

Eden izmed večjih delov modela sistema, ki ni del UML specifikacije, je model uporabniškega vmesnika. Pomemben je zato, ker predstavlja predviden izgled sistema z vidika končnega uporabnika. Odvisen je od namena sistema, uporabljenih tehnologij ter ciljnega končnega uporabnika.
Model je lahko v papirnati obliki zaslonskih mask, lahko pa tudi v bolj naprednih računalniških simulacijah, ki se omejeno odzivajo na uporabnika. Tu že govorimo o prototipu uporabniškega vmesnika. Tako prototipiranje je v zadnjem času izredno priljubljeno. Za ta namen so nastala mnoga napredna orodja, ki simulirajo delovanje uporabniških vmesnikov.

1.4 Vpliv metodologije razvoja na modeliranje

Metodologije razvoja programske opreme določajo načine dela, po katerih nastane sistem. Različnih izvedenk metodologij je veliko, delijo pa se na naslednje skupine [16]:

- Agilne metodologije
- Inkrementalne metodologije
- Prototipne metodologije
- Slapovne metodologije
- Spiralne metodologije

Metodologija dela med drugim posredno določa tudi model sistema. Slapovni model sistema predvideva fazo analize in fazo načrtovanja pred fazo izdelave. To pomeni, da je vsa analiza sistema izdelana pred izdelavo sistema, kar zahteva izjemno podrobn model sistema, saj se glede na tega kasneje izdela cel sistem.
Agilne metodologije dela delujejo na drugačnih načelih. Faze analize in izdelave se nenehno izmenjujejo. Posamezna faza je manj obsežna. Razvijalci zasnujejo in izdelajo nek majhen del sistema, preden nadaljujejo z drugim. Pri takem načinu izdelave je model še vedno pomemben, saj skrbi za celovitost rešitve ob koncu projekta. Potreba po zelo podrobnem modelu je manjša, saj je krovni model manj podroben, posamezne dele sistema pa se modelira po potrebi. Končni sistem je v mnogo primerih drugačen od začetnega načrta, saj se sistem tekom izdelave nekoliko spreminja glede na predloge, izkušnje ter izive.
Na modeliranje še najbolj vpliva tip uporabljene metodologije. Lahke metodologije (npr. agilne metodologije) ne predpisujejo modelov, ali pa predpisujejo manj podrobne modele.
Težke metodologije so podrobnejše. Določujejo posamezne potrebne modele, diagrame in tehnična besedila. Primer take metodologije je tudi t.i. Rational Unified Proces - RUP.

1.4.1 Rational Unified Process

Rational Unified Proess (RUP) je iterativna metodologije razvoja programske opreme. Zgrajena je modularno in prilagodljiva za različne tipe ter velikosti

projektov. Razvila se je poleg standarda UML.
RUP je zgrajen na več vsebinskih sklopih oz. modulih. Vsak opisuje navodila, potrebne veščine ter izdelke za zaključek modula.
RUP opisuje 4 glavne faze razvoja projekta: vzpostavtev projekta, zbiranje informacij, konstrukcija ter uvedba in prevzem.
V fazi zbiranja informacij RUP podrobno definira tudi strukturo zajema zahtev. Ta vključuje popis obstoječih poslovnih procesov, funkcionalne zahteve, nefunkcionalne zahteve ter diagram primerov uporabe. Prav diagram primerov uporabe je nato podlaga za opise primerov uporabe, na katerih se nato gradi podroben opis sistema z modelom uporabniškega vmesnika, ER podatkovnim modelom ter razrednim modelom, značilnim za objektni pristop programiranja [14].
V tej nalogi bomo tako upoštevali RUP metologijo ter z zajemom zahtev opisali potrebe študentskih klubov za podporo poslovanju pred implementacijo sistema.

Poglavje 2

Metoda dela in uporabljena orodja

2.1 Metoda dela

Model informacijskega sistema se lahko izdela pred implementacijo sistema ali po njej.
V prvem primeru sistem modeliramo pred izvedo. To nam omogoča svobodo pri modeliranju, saj lahko različne zahteve ali poslovne procese realiziramo na več načinov.
V drugem primeru gre za analizo sistema. Tu dejanski sistem analiziramo z nalogo boljšega razumevanja ter dokumentiranja delovanja sistema. Ponavadi se taka analiza opravi, ko delovanja obstoječega sistema ne razumemo, ali pa se pripravljamo na prenovo [13].
Za potrebe te naloge smo se odločili za modeliranje sistema pred izvedbo, saj je namen naloge rešiti morebitne probleme, ki nastanejo pri implementaciji sistema glede na predhodni model ter popisati postopek uveljavitve modela. Tako bomo najprej izdelali model sistema, nato pa bomo sistem zgradili skladno z modelom. Opisali bomo morebitne razlike in izzive v izdelavi sistema glede na predhodni model.

2.2 Uporabljena orodja in tehnologije

Modeliranje in opis sistema se lahko opravi s pomčjo več različnih orodij. Specifikaciji UML ali RUP ne predpisujeta specifičnega orodja za modeliranje oz. izvedbo sistema.
Tehnologije, ki so izbrane v procesu zajema zahtev, pa kasneje vplivajo tako na modele kot kasneje na samo implementacijo zaradi svojih specifik in omejitev. Čeprav na začetku izdelave sistema točne tehnologije pogosto še niso izbrane, saj je to del predhodne analize, bodo orodja in tehnologije, ki bodo kasneje uporabljene, opisane v tem delu naloge.
Izbira orodij je temeljila na kriterijih, ki so opisani v poglavju 3.1.7 Nefunkcionalne zahteve. Večina izbranih orodij je odprtokodnih in široko uporabljenih. Zanje je značilno, da se redno posodabljajo, so dobro dokumentirana, prav tako pa je zaradi razširjenosti ustvarjena velika skupnost razvijalcev, ki odgovarja na vprašanja ter rešuje pogoste težave.

2.2.1 Zajem zahtev in modeliranje

Za osnovni zajem zahtev so bila uporabljena pisarniška orodja Microsoft Office, vendar so bili kasneje modeli prenešeni v bolj formalno obliko z naslednjimi orodji:

Lucid Chart

Za izdelavo vseh UML modelov ter modela uporabniškega vmesnika pa tudi drugih podobnih modelov v tej nalogi, je bilo uporabljeno spletno orodje Lucid Chart. Spletna aplikacija podpira mnogo različnih standardov, med drugim tudi UML. Uporabnik prek pred pripravljenih knjižic izbira elemente, jih postavlja ter povezije v diagrame in skice. Orodje omogoča uvoz iz drugih popularnih orodij, kot na primer Microsoft Visio, izvaža pa v več dokumentnih ter slikovnih formatih. Omogoča tudi delo v skupini.

2.2.2 Spletni odjemalec

Vsaka spletna aplikacija sestoji iz dveh ločenih delov: spletnega strežnika in spletnega odjemalca. Spletni odjemalec navadno poganja del aplikacije, ki je zadolžena za prikaz spletne strani, lahko pa tudi del aplikacije, ki je zadolžena za logiko spletne aplikacije.
V primeru Kurnika bo spletni odjemalec realiziran z pomočjo knjižice AngularJS.

AngularJS

AngularJS je prva iteracija odprtokodnega ogrodja za izdelavo spletnih aplikacij, ki temelji na programskem jeziku JavaScript. Namenjena je izdelavi enostranskih spletnih aplikacij. Implementira t.i. Model View Controler - MVC arhitekturo [9].

2.2.3 Google Material Design

Google Material Design je specifikacija, ki definira namen, izgled, obnašanje ter spreminjanje posameznih elementov uporabniškega vmesnika. Namen specifikacije je izdelava intuitivnih in razumljivih uporabniških vmesnikov.
Specifikacija ne predpisuje podrobnega izgleda gumbov, posamezne pisave ali določene barvne kombinacije. Osredotoči se raje na namen posameznih elementov glede na njihovo pozicijo, obliko in obnašanje. Google ne izdaja knjižic za implementacijo specifikacije, temveč to prepusi odprtokodnim projektom ter posameznim razvijalcem.
Cilj specifikacije je podati orodje, ki pomaga ustvariti lep enoten in razumljiv uporabniški vmensik ter omogoči dobro uporabniško izkušnjo vse od majhnega zaslona mobilnega telefona do velikega zaslona računalnika.
Kurnik zaradi uporabe aplikacijske platforme AngularJS za implementacijo Material Design-a uporablja knjižnico AngularJS Material. Knjižnica implementira večino elementov uporabniškega vmesnika, ki jih predpisuje specifikacija Material Design. Prednost uporabe take knjižnice na spletnem odje-

malcu je v lažji izdelavi uporabniškega vmesnika ter enotni podobi, ki jo ima končna aplikacija [1].

2.2.4 Zaledni sistem

Zaledni sistem vsebuje del spletne aplikacije, ki teče na strežniku. Streže zahtevam vseh instanc spletnega odjemalca oz. osprednega dela spletne aplikacije.
Zaledni sistem ponavadi vključuje hrambo podatkov. Tako je tudi v primeru sistema Kurnik. V drugih primerih lahko zaledni sistem opravlja tudi vse logične operacije ter na zahtevo zgradi spletno stran preden jo streže spletnemu odjemalcu.
Za potrebe spletnega strežnika bomo uporabljali okolje Node.js ter spletni strežnik Express, ki teče v njem. Poleg spletnega strežnika bomo v spletni aplikaciji Kurnik uporabljali tudi podatkovno bazo MySQL, ki bo skrbela za hrambo podatkov.

Node.js

Node.js je priljubljena strežniška tehnologija za izdelavo zalednih sistemov. Izvaja programsko kodo napisano v jeziku JavaScript. Tehnologija teče na vseh večjih operacijskih sistemih. To olajša razvoj, saj je mogoče razvijati zaledne sisteme na namiznih računalnikih, nato pa zaledni sistem brez večjh sprememb prenesti na za to namenjen strežnik [5].

Express

Express je popularno ogrodje namenjeno izdelavi spletnih aplikacij skupaj s tehnologijo Node.js. Zaradi enostavnosti in široke uporabe je postal de-facto spletni strežnik za projekte, ki za zaledni sistem uporabljajo Node.js [2].

MySQL

Za hrambo podatkov skrbi relacijska podatkovna baza Oracle MySQL. Ta odprtokodni sistem za hrambo podatkov je široko razširjen. Poleg tega lahko deluje na širokem naboru operacijskih sistemov [4].

Izbrana orodja tečejo na širokem naboru operacijskih sistemov. To olajša razvoj, saj lahko razvijalec razvija v svojem okolju, ki je neodvisno od ciljnega strežnka.

Poglavje 3

Model informacijskega sistema Kurnik

Model informacijskega sistema Kurnik je izdelan po RUP metologiji ter obsega naslednje podmodele:

- Zajem zahtev
- Model podatkovne baze
- Model uporabniškega vmesnika

Pred vsakim modelom je nekaj uvodnih besed o namenu modela, nato pa je predstavljena vsebina posamezenega modela za informacijski sistem Kurnik.

3.1 Zajem zahtev

Zajem zahtev je prva aktivnost, ki se odvije v procesu ustvarjanja modela sistema. RUP metologija predvideva začetno fazo, kjer se rešujejo naslednja vprašanja [14]:

- Razumeti, kaj izdelati
- Identificirati ključne funkcionalnosti sistema

- Predložiti vsaj eno možno rešitev
- Razumeti stroške, časovnico ter tveganja povezana z projektom
- Določiti procese, orodja in tehnike, ki bodo uporabljene

Za doseganje zgoraj naštetih ciljev sta Wiggers in Beatty definirala naslednje možne tehnike[16]:

- Intervjuji
- Delavnice
- Fokusne skupine
- Opazovanja
- Vprašalniki
- Analize sistemskih in uporabniških vmesnikov
- Analiza uporabljenih dokumentov

V procesu pridobivanja podatkov za Kurnik so bili uporabljeni intervjuji, fokusne skupine, vprašalniki in analiza dokumentov, ki se uporabljajo.

3.1.1 Opis problemske domene

Opis problemske domene je uvodni opis problema, ki bralcu modela sistema poda začetni kontekst.
Študentski klub (ŠK) je klub študentov ter dijakov. Deluje pod okriljem Slovenske Študentske Organizacije (ŠOS). Svojim članom nudi različne ugodnosti, organizira dogodke različnih vrst ter si prizadeva za izboljšanje prostega časa ter kvalitete življenja študentov in dijakov. Delovanje študentskega kluba določajo predpisi ŠOS.

3.1.2 Slovar izrazov

Sledi izdelava slovarja izrazov. Tu so pojasnjeni izrazi, ki se uporabljajo v zajemu zahtev in so specifični za problemsko domeno. Sem spadajo tudi posebne fraze, besedne zveze ali drugi izrazi, ki imajo v predstavljeni problemski domeni drugačen pomen kot sicer.

- **Študentski klub (ŠK)** - Klub študentov in dijakov, ki si prizadeva za boljšo kvaliteto življenja svojih članov z svojimi aktivnostmi
- **Član študentskega kluba (član)** - Študent ali dijak z veljavnim potrdilom o izobraževanju (trenutno redno obiskuje izbraževalni program v RS), ki se včlani v študentski klub
- **Klubski status oz. članstvo** - Status člana v klubu
- **Aktivist študentskega kluba (aktivist)** - Delavec ŠK, ki preko urejenega delovnega razmerja ali prostovoljno deluje v ŠK
- **Kurnik** - Informacijski sistem za podporo poslovanju študentskih klubov

3.1.3 Popis obstoječih poslovnih procesov

Popis obstoječih poslovnih procesov ni del specifikacije zahtev, kakor jo predpisuje RUP. Ta neformalen opis trenutnih poslovnih procesov je uporaben pri nadaljnjem razumevanju problemske domene. Spisan je v naravnem jeziku. Popis obstoječih poslovnih procesov je pogosto tudi podlaga za funkcionalne zahteve ter primere uporabe, še posebej če se poslovne procese le digitalizira oz. se jih z pomočjo informacijskih sistemov posodobi. Tako je tudi pri sistemu Kurnik, kjer gre v večini za podporo obstoječih poslovnih procesov [16].

Članstvo

V študentski klub se lahko vpiše vsaka fizična oseba, vendar je klub primarno namenjen študentom in dijakom. V neki upravni enoti lahko deluje le en študentski klub. Člani imajo pravico do včlanitve (in pridobitve ugodnosti) le v tistem študentskem klubu, kjer imajo prijavljeno stalno prebivališče. Članstvo je lahko:

- **Aktivno:** Član ŠK se trenutno izobražuje v srednji šoli ali višješolski ustanovi
- **Zamrznjeno:** Član se je izobraževal v preteklem šolskem letu
- **Zapadlo:** Kadar član ne spada v zgornje kategorije.

Študent ali dijak se lahko včlani v ŠK ob uradnih urah ŠK. Z osebnim dokumentom ter originalnim potrdilom o šolanju aktivist preveri istovetnost in status bodočega člana. Bodoči član izpolni prijavni list. Na njem poda svoje ime, priimek, naslov, telefon ali e-naslov, soglasje za prejemanje e-novic, izobraževalno ustanovo, program in letnik ter katere izmed kategorij aktivnosti ga zanimajo (šport, kultura, izobraževanje, zabava). Aktivist ter študent oz. dijak ga podpišeta. Aktivist prijavni list fotokopira, članu izroči kopijo kot pordilo o včlanitni v ŠK nato pa v evidence ŠK vstavi novo vpisnico skupaj s kopijo potrdila o šolanju.
Član lahko nato koristi različne ugodnosti, ki jih nudi študentski klub. Status v ŠK velja do konca tekočega šolskega leta.
Član podaljša oz. obnovi aktivno članstvo tako, da ob uradnih urah aktivistu preda novo potrdilo o šolanju ter osebni dokument za identifikacijo. Nadaljnji postopek je enak kakor ob vpisu.

Dogodki

Študentski klub tekom svojega dela organizira različne dogodke. Dogodki se delijo na športne, izobraževalne, družabne ter zabavne. Imajo lahko prijavnino, ki je lahko različna za aktivne, zamrznjene ter zapadle člane ŠK.

Dogodek je lahko tudi številčno omejen (recimo največ prostih mest na avtobusu). Nov dogodek uskladi izvršni odbor ŠK. Na njem se uskladi tema dogodka, ime, kraj in čas dogajanja, plačilo, možne omejitve ipd.
Član se prijavi na dogodek med uradnimi urami ŠK. Aktivistu predstavi osebni dokument, po potrebi plača prijavnino, aktivist pa članu preda potrdilo o prijavi ter račun, če je dogodek plačljiv.
Član se lahko pred dogodkom tudi odjavi. V primeru plačljivega dogodka mu pripada vračilo prijavnine, v vsakem primeru pa potrdilo o odjavi.

Sporočanje

ŠK preko e-pošte sporoča novice ter druge informacije svojim članom. Sporočanje je namenjeno:

- Posamezniku
- Ciljni skupini prejemnikov (udeleženci dogodka, moški, ženske, dijaki, študenti, zaintersiranim za neko tematiko)
- Vsem članom ŠK

3.1.4 Opisi primerov uporabe

V načrtovanju kateregakoli programa oz. sistema je najprej potrebno razumeti, kaj bodoči uporabniki potrebujejo in kaj želijo doseči z sistemom. Medtem ko je mogoče sistem zasnovati okoli planiranih funkcionalnosti sistema ter kasneje upati (oz. načrtovati), da bodo le te pokrile potrebe uporabnikov, je veliko bolje, da sistem načrtujemo iz uporabniškega zornega kota. Osredotočenje na uporabnika ter njegovo rabo sistema zagotovi, da ne pride do razvoja nepotrebnih funkcionalnosti sistema, hkrati pa pomaga pri razvrščanju prednosti potrebnih funkcij[16].
Tako Wiggers kot Fowler opisujeta primer uporabe kot opise tipičnih interakcij med uporabnikom sistema in samim sistemom skupaj z opisom okoliščin, v katerih je sistem takrat uporabljen[11][13]. Wiggers prav tako dodaja, da

primeri uporabe ponavadi sestojijo iz povedka in osebka (npr. urejanje dogodka).
Ime samega primera uporabe je sestavljeno tako, da samo po sebi opisuje to dejanje. Opis posameznega primera uporabe pa poda posamezne korake dejanja ter z njimi podrobnosti in kontekst uporabe sistema. Tak način je prikazan spodaj v opisih primerov uporabe za sistem Kurnik.
Drug pogost način opisov primerov uporabe so uporabniške zgodbe (angl. user stories). To so kratki opisi posamezne funkcionalnosti sistema z zornega kota uporabnika sistema ali stranke, ki se pogosto uporabjajo v agilnih metodologijah dela. Po navadi sledijo naslednjemu receptu:

Kot uporabik sistema želim izvesti poljubno akcijo, da dosežem pripadajoč cilj.

Uporabniške zgodbe se lahko tudi kombinirajo z osnovnimi primeri uporabe.
Opisi primerov uporabe oz. uporabniške zahteve v hierarhičnem spektru zahtev ležijo med poslovnimi zahtevami, ki so delno definirane v opisu problemske domene ter funkcionalnimi zahtevami, ki natančno opisujejo potrebne funkcionalnosti za implementacijo.
Načrtovalci programske opreme že dolgo uporabljajo uporabniške scenarije kot način opisa primerov uporabe [10]. Uporabniške scenarije so prav tako posvojile moderne agilne metodologije razvoja programske opreme.
Spodaj so opisani identificirani primeri uporabe za problemsko domeno študentskega kluba. Opisi bodo služili kot podlaga za izdelavo modela primerov uporabe ter definicijo podrobnih funkcionalnih in nefunkcionalnih zahtev.
Sestava opisa primera uporabe sestoji iz:

1. Cilj: Cilj oz. cilji primera uporabe.

2. Predpogoj: Vsi predpogoji, ki morajo biti izpolnjeni za začetek primera uporabe.

3. Glavni potek: Predvideni idealni potek primera uporabe.

4. Razširitve: Vsi predvidljivi možni poteki primera uporabe, ki se od predvidenega poteka razlikujejo v navedeni točki.

Prijava aktivista ali administratorja v sistem

Cilj: Prijava aktivista ali administratorja v sistem Kurnik.
Predpogoj: Aktivist ali administrator je vnešen v sistem.

1. Aktivist z spletnim brskalnikom dostopa do spletne strani spletnega sistema.
2. Aktivist vnese svoje uporabniško ime in geslo.
3. Sistem odobri prijavo ter izvede preusmeritev spletnega brskalnika na domačo stran spletnega sistema.

Razširitve:
2a: Aktivist vnese napačno uporabniško ime in geslo:

1. Sistem izpiše sporočilo:

   Napačno uporabniško ime ali geslo.

Vpis člana v ŠK

Cilj: Vpis fizične osebe v študentski klub.
Predpogoj: Oseba še ni član ŠK.
Glavni potek:

1. Bodoči član pristopi k aktivistu.
2. Bodoči član predloži veljavno osebno izkaznico ter potrdilo o šolanju.
3. Aktivist potrdilo o šolanju fotokopira.

4. Aktivist izpolni prijavo člana. Vpiše osebne podatke, podatke o izobrazbi ter podatke o zanimanjih člana.
5. Aktivist natisne in podpiše 2 prijavna lista.
6. Član podpiše obe kopiji. Eno zadrži kot potrdilo o vpisu v ŠK.
7. Aktivist prijavni list in potrdilo o vpisu skenira in naloži v sistem.

Razširitve:
2a: Osebna izkaznica ni veljavna:

1. Aktivist zavrne prošnjo za podaljšanje statusa ter konča postopek.

2b: Potrdilo o šolanju ni veljavno:

1. Aktivist nadaljuje z vpisom, vendar člana vpiše brez izobrazbe. To bo povzročilo, da bo član vpisan s statusom zapadlo. Glej Glavni potek vpisa.

6a: Član zavrne podpis prijavnice:

1. Aktivist zavrne prošnjo za podaljšanje statusa ter konča postopek.

Urejanje podatkov člana ŠK

Cilj: Sprememba podatkov člana ŠK.
Predpogoj: Oseba je član ŠK.
Glavni potek:

1. Član ŠK pristopi k aktivistu
2. Član predloži veljavno osebno izkaznico.
3. Aktivist preveri veljavnost osebne izkaznice ter najde člana v sistemu.
4. Aktivist spremeni podatke člana tako, da izbere možnost 'Uredi' v sistemu.

5. Sistem natisne potrdilo o spremembi podatkov.

6. Aktivist preda potrdilo o spremembi podatkov v sistemu.

Razširitve:
3a: Osebna izkaznica ni veljavna:

1. Aktivist zavrne prošnjo za podaljšanje statusa ter konča postopek.

Podaljšanje statusa člana ŠK

Cilj: Podaljšanje aktivnega statusa člana.
Predpogoj: Član se v tekočem šolskem letu izobražuje na izobraževalni ustanovi (srednja šola, visoka šola, višja šola).
Glavni potek:

1. Član pristopi k aktivistu.

2. Član predloži veljavno osebno izkaznico ter potrdilo o šolanju.

3. Aktivist potrdilo o šolanju fotokopira.

4. Aktivist najde izbere možnost "Člani"ter najde člana preko iskalnega polja ali tabele.

5. Aktivist izbere možnost"podaljšaj status" ter izpolni podatke o novem izobraževanju člana.

6. Aktivist natisne in podpiše potrdilo o spremembi podatkov.

7. Član podpiše potrdilo o spremembi podatkov.

8. Aktivist v sistem naloži skenirano potrdilo o šolanju.

Razširitve:
2a: Osebna izkaznica ni veljavna:

1. Aktivist zavrne prošnjo za podaljšanje statusa ter konča postopek.

6.a: Član zavrne podpis prijavnice:

1. Aktivist zavrne prošnjo za podaljšanje statusa ter konča postopek.

Izpis člana iz ŠK

Cilj: Izpis člana iz ŠK
Predpogoj: Oseba je član ŠK
Glavni potek procesa:

1. Član pristopi k aktivistu.
2. Član predloži veljavno osebno izkaznico.
3. Aktivist izbere možnost "člani"ter najde člana.
4. Aktivist izbere možnost "zbriši člana" ter potrdi izbiro.
5. Sistem natisne potrdilo o izpisu člana iz ŠK.
6. Aktivist članu preda natisnjeno potrdilo o izbrisu člana.

Razširitve:
2a: Osebna izkaznica ni veljavna:

1. Aktivist zavrne prošnjo za izpis člana ter konča postopek.

Ustvarjanje novega dogodka

Cilj: Ustvariti nov dogodek v sistemu.
Predpogoj: Dogodek z enakim imenom ne obstaja.
Glavni potek procesa:

1. Aktivist izbere "Dogodki" v sistemu.
2. Aktivist izbere možnost "Ustvari nov dogodek".
3. Aktivist vpiše podatke o dogodku.
4. Aktivist potrdi nov dogodek.

Prijava člana na dogodek

Cilj: Prijaviti člana na dogodek.
Predpogoj: Član ni prijavljen na dogodek.
Glavni potek procesa:

1. Član pristopi k aktivistu.
2. Aktivist najde dogodek v sistemu prek tabele ali iskalnega polja.
3. Aktivist izbere “Dodaj prijavo”.
4. Aktivist preko iskalnika najde člana ter potrdi prijavo.
5. Aktivist potrdi plačilo, če je dogodek plačljiv in član plača dogodek.
6. Sistem natisne potrdilo o prijavi.
7. Aktivist natisnjeno potrdilo o prijavi preda članu.

Razširitve:
5a: Član zavrne plačilo dogodka:

1. Aktivist prekine postopek ter izbriše člana z seznama udeležencev dogodka.

Odjava člana z dogodka

Cilj: Odjaviti člana z dogodka
Predpogoj: Član je prijavljen na dogodek
Glavni potek procesa:

1. Član pristopi k aktivistu.
2. Aktivist izbere ”dogodki”.
3. Aktivist najde dogodek v sistemu ter izbere seznam prijavljenih članov.
4. Aktivist preko iskalnika oseb ter seznama označi prijavljeno osebo ter izbere možnost “odjava člana”.

5. Aktivist potrdi odjavo.
6. Sistem natisne potrdilo o odjavi.
7. Aktivist vrne plačilo dogodka, če je ta plačljiv ter članu izroči plačilo dogodka.

Sprememba podatkov dogodka

Cilj: Sprememba podatkov dogodka.
Predpogoj: Dogodek je ustvarjen v sistemu.
Glavni potek procesa:

1. Aktivist najde dogodek na seznamu dogodkov ter ga izbere.
2. Aktivist izbere možnost 'Uredi' ter izvede željene spremembe.
3. Aktivist shrani spremembe.

Izbris dogodka

Cilj: Izbris dogodka iz sistema.
Predpogoj: Dogodek je v sistemu.
Popogoj: Dogodka ni v sistemu. Glavni potek procesa:

1. Aktivist najde dogodek na seznamu dogodkov ter ga izbere.
2. Aktivist izbere možnost 'Izbriši' ter izvede željene spremembe.
3. Aktivist potrdi izbris.

Pregled poslanih sporočil

Cilj: Pregled že poslanih sporočil.
Predpogoj: Vsaj eno sporočilo je bilo predhodno poslano z uporabo sistema.
Glavni potek procesa:

1. Aktivist izbere možnost “Sporočila”.
2. Aktivist najde sporočilo prek seznama ali iskalnega polja ter ga izbere.
3. Sistem prikaže vsebino ter naslovnike sporočila.

Pošiljanje sporočil članom ŠK

Cilj: Ciljnim članom ŠK poslati e-sporočilo
Predpogoj: Vsi prejemniki so kot člani prisotni v sistemu ter imajo prisoten podatek o e-poštnem naslovu. Glavni potek procesa:

1. Aktivist izbere sekcijo “sporočanje”.
2. Aktivist najde naslovnika ali skupino naslovnikov. To so lahko udeleženci dogodkov, dijaki, študenti, člani, ki jih zanima določena kategorija dogodkov ali spol.
3. Aktivist vnese sporočilo.
4. Aktivist potrdi odjavo.
5. Aktivist izbere možnost “pošlji”.
6. Sistem izpiše sporočilo ob uspešnem poslanem sporočilu.

Razširitve:
5a: Sistem izpiše sporočilo o napaki

1. Sistem pozove aktivista naj kontaktira administratorja.

Vpis novih aktivistov v sistem

Cilj: Ustvariti novega aktivista v sistemu Kurnik.
Predpogoj: Aktivist še nima profila v sistemu Kurnik. Glavni potek procesa:

1. Administrator izbere možnost “Administracija sistema”.
2. Administrator izbere možnost “Dodaj novega aktivista”.

3. Administrator vnese podatke o aktivistu, (Ime, priimek, rojstni podatki, naslov, e-poštni naslov, telefonska številka).

4. Administrator potrdi ustvarjanje novega aktivista.

5. Sistem pošlje e-poštno obvestilo na naslov dan v (4.) o novem profilu v tem sistemu, ter ga poziva naj potrdi svoj novi profil.

6. Aktivist potrdi novi profil, ga dokončno ustvari ter vnese svoje geslo s povezavo v e-sporočilu.

Razširitve:
6a. Aktivist ne portdi svojega profila

1. Po 60 dneh profil zastara in se izbriše.

Urejanje podatkov profila aktivista

Cilj: Urediti podatke aktivista v sistemu Kurnik (osebni profil).
Predpogoj: Aktivist ima ustvarjen profil ter je vanj prijavljen kot aktivist ali administrator. Glavni potek procesa:

1. Aktivist izbere svoj profil.

2. Aktivist izbere možnost “Uredi”.

3. Aktivist uredi željene podatke.

4. Aktivist potrdi nove podatke.

Izbris profila aktivista

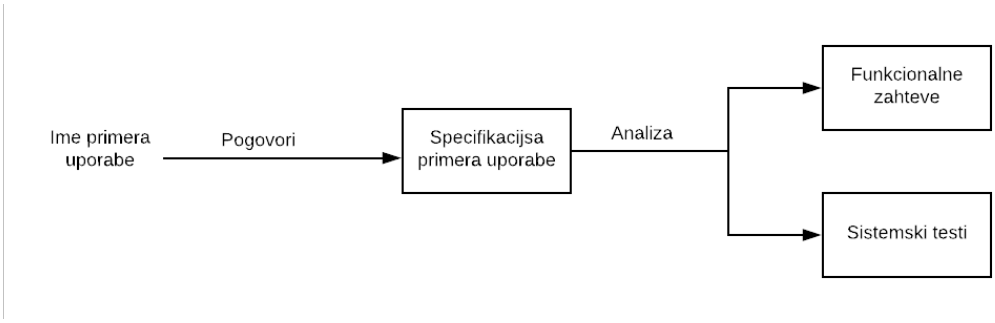
Cilj: Izbris profila aktivista iz sistema Kurnik.
Predpogoj: Aktivist je vnešen v sistem. Glavni potek procesa:

1. Administrator izbere možnost “administracija aktivistov” iz menija.

2. Administrator označi željenega aktivista iz seznama vseh aktivistov.

3. Administrator izbere možnost “izbriši” ter potrdi izbris.

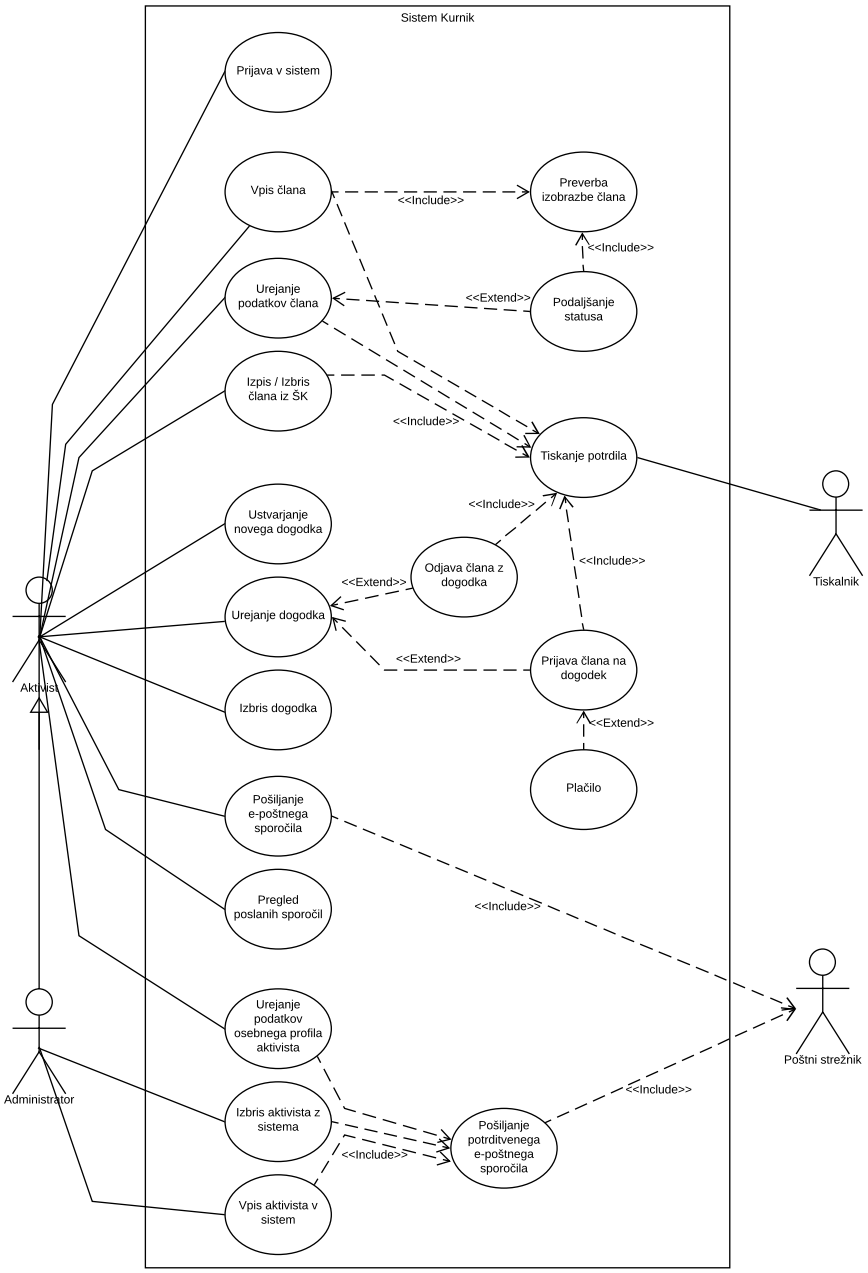
Opisani primeri uporabe so pri agilnih metodologijah končna stopnja formalnega modela informacijskega sistema, in služijo kot nastavki za kasnejše pogovore med stranko, sistemski načrtovalci ter programerji. Pogovori se zgodijo tik pred razvojem ter po potrebi. Odkrijejo dodatne informacije, ki jih mora razvojna ekipa upoštevati pri razvoju sistema. Primeri uporabe, ki so preveliki za zgodbo v agilnem pristopu dela, se na podlagi teh pogovorov po potrebi razbijejo na manjše zgodbe, ki so potrebne za implementacijo [16]. Naš primer bo segal dlje. Primeri uporabe bodo služili kot podlaga za končne funkcionalne zahteve sistema. Prav tako je to primarni vir specifkacije sistemskih testov, ki pa ni del te analize.



Slika 3.1: Pretvorba uporabniških zahtev v funkcionalne zahteve

Diagram primerov uporabe

Na podlagi opisov primerov uporabe lahko izdelamo diagram primerov uporabe (angl. use case diagram). Diagramska tehnika je standardizirana ter del UML specifikacije, ki določa elemente na diagramu. Z diagrama 3.1.4 je razvidno, da gre za 2 profila uporabnikov: aktivista in administratorja, ki prek različnih primerov uporabe upravljata s člani študentskega kluba. Z diagrama je prav tako razvidno, da je identificiranih 11 primerov uporabe ter 7 razširitev. Te so označene z oznakami <<extend >>, če je razširitev



Slika 3.2: Diagram primerov uporabe

opcijska oz. <<include >>, če je razširitev obvezna ter se zgodi tekom primera uporabe.
Diagram 3.1.4 prav tako nakazuje vrstni red poteka primera uporabe. Z uporabo takega modela ter opisa primera uporabe lahko kasneje funkcionalnost učinkovito podpremo v spletni aplikaciji. Bolj zapleteni primeri uporabe in posledično zapletene funkcionalnsoti lahko vključujejo tudi druge diagramske tehnike kot so odločitvena drevesa, diagrami podatkovnih tokov, diagrami aktivnosti ali diagrami poslovnega procesa.

Poslovna pravila

Vsaka organizacija posluje v skladu z različnimi zakoni, nabori pravil in dogovori v organizaciji. Na primer: Član študentskega kluba je lahko vsaka fizična oseba s stalnim prebivališčem v RS, vendar bo ugodnosti ŠK deležna le oseba, ki je vpisana v študentski klub ter obiskuje srednješolsko ali višješolsko izobrazbo.
Poslovna pravila se v večini primerov izoblikujejo zunaj konteksta neke specifične aplikacije ali informacijskega sistema. Tudi če določenega informacijekga sistema ne bi bilo, bi večina poslovnih pravil obstala. Tako je potrebno informcaijski sistem prilagoditi poslovnim pravilom organizacije ter jih spoštovati.
Mnogo poslovnih pravil je implicitno opisanih v opisih primerov uporabe, vendar se hitro pripeti, da se kako poslovno pravilo izgubi oz. se interpretira napačno. Zato je dobra praksa, da se poslovna pravila zapiše in potrdi pred izvedbo informacijkega sistema.
Za lažjo referenco kasneje v različnih dokumentih je dobra praksa, da različna poslovna pravila tudi kodiramo oz. jih drugače enolično označimo[16]. V našem primeru smo različna poslovna pravila označili s ŠK ter zaporedno številko poslovnega pravila.

- **ŠK1: Organizacija dela:** Študentski klub je organiziran kot neprofitna organizacija. Njegovo delovanje določa zveza študentskih klubov slovenije - ŠOS.

- **ŠK2: Stalno prebivališče člana ŠK:** V ŠK se lahko vpšiejo le člani iz upravne enote, kjer deluje ŠK. Oseba stalno prebivališče dokaže z predložitvijo osebnega dokumenta.
- **ŠK3: Status člana v ŠK:** Študenti in dijaki v ŠK imajo lahko aktivni, zamrznjeni ali zapadli status. Aktivni status pridobijo ob vpisu v ŠK z tekočo srednješolsko ali visokošolsko izobrazbo. Ta v naslednjem šolskem letu postane zamrznjen oz. neaktiven.
- **ŠK4: Koriščenje ugodnosti:** Član ŠK lahko koristi ugodnosti ŠK (nižje cene dogodkov ter kuponov), če ima aktiven oz. zamrznjen status.
- **ŠK5: Vpis aktivistov na dogodek:** Dogodki imajo lahko omejeno število prijav. V tem primeru se je možno prijaviti do zapolnitve mest.
- **ŠK6: Podaljšanje statusa:** Član lahko podaljša status tako, da predloži veljavno potrdilo o vpisu ter izpolni novo vpisnico.
- **ŠK7: Hramba dokumentov:** ŠK hrani vse dokumente povezane s člani najmanj 5 let.
- **ŠK8: Odgovorna oseba:** Za delovanje kluba je odgovorem predsednik kluba, ki ga imenuje izvršni odbor kluba.

Klub temu, da je poslovnih pravil veliko, se mnogokrat izkaže, da je za implementacijo sistema potrebno upoštevati le nekaj poslovnih pravil zaradi omejitev sistema. Tako na primer ni potrebno upoštevati pravil ŠK1 ter ŠK8, saj sta zunaj mej našega informacijskega sistema.

3.1.5 Funkcionalne zahteve

Mnogi razvijalci opisane primere uporabe predstavijo kakor končne funcionalne zahteve. Vendar so primeri uporabe le uporabnikov zorni kot uporabe sistema. Ne vsebujejo vseh informacij, ki so potrebne za razvoj samega sistema.

Uporabnik bankomata ne ve, ali ob dvigu denarja poteka kakšna komunikacija med banko in bankomatom. Take podrobnosti so nevidne uporabnikom, vendar so nujno potrebne za delovanje sistema.
To je razumljivo, vendar se tu lahko vprašamo, kaj vse obsegajo kvalitetne funkcionalne zahteve, ki celostno opisujejo nek informacijski sistem?
Funkcionalne zahteve obsegajo vse funkcije sistema ter jih opisujejo do take mere, da je zanje moč razviti neko rešitev. Vsako zahtevano funkcijo je potrebno podrobno opisati. Pri tem lahko uporabljamo naslednje tehnike:

- Diagrami podatkovnih tokov
- Procesni diagrami
- Diagrami prehajanj stanj
- Odločitvena drevesa ter tabele
- Diagram primerov uporabe
- Diagrami aktivnosti
- Diagrami entiteta-razmerje
- Naravni jezik
- Strukturirani jezik

Po Wiggersu naj bi bile dobre funkcionalne zahteve [16]:

- **Popolne:** Vsaka zahteva naj bi vsebovala vse informacije za dobro razumevanje bralca. To pomeni, da bi iz funkcionalne zahteve razvijalec funkcionalnost implementiral brez ugibanj ali težav.
- **Pravilne:** Vsaka zahteva naj bi zadovoljila neko potrebo uporabnika sistema. Vsaka zahteva bi morala biti sledljiva do začetnega vzroka zahteve - poslovnega pravila, primera uporabe ali drugega razloga za zahtevo. Zahteve se medsebojno ne smejo negirati.

- **Izvedljive:** Zahteve naj bi bile tudi izvedljive upoštevajoč omejitve ter zmogljivosti tehnologij sistema kot tudi omejitve projekta, časovnice ter cenovnega okvirja. V procesu zbiranja zahtev lahko sodelujemo z razvojniki, ki podajo oceno o izvedljivosti dane zahteve.

- **Potrebne:** Vsaka zahteva naj bi vsebovala funkcijo, ki deležniku prinese dodano vrednost uporabe sistema. Alternativno je zahteva lahko nujna za implementacijo zaradi poslovnih pravil, različnih zakonov ali standardov.

- **Prioritizirane:** Vsaka zahteva naj bi imela oznako prioritete razvoja glede na potrebe stranke ter potreben vložek dela. Vsaki zahtevi pripišemo pomembnost, saj lahko tako hitreje zagotovimo delovanje nujnih funkcionalnosti sistema.

- **Nedvoumne:** Naravni jezik je podvržen dvoumnosti. Razvijalec lahko neko zahtevo zapisano v naravnem jeziku razume drugače kot načrtovalec sistema. Dvoumnost se lahko pojavi tudi v razlikah med interpretacijo funkcionalnosti stranke ter izdelovalca sistema.

- **Preverljive:** Funkcionalnosti naj bi bile preverljive tako, da tretja oseba z danim načrtom testiranja preveri in potrdi izvedbo funkcionalnosti.

Objektni pristop k razvoju programske opreme velik del sistema že predstavi v opisu primerov uporabe. Zato lahko funkcionalne zahteve zgradimo na naslednje načine [16]:

- **Zgolj opisi primerov uporabe:** Ena možnost je, da opise primerov uporabe le dopolnimo z opisi funkcij, kjer obstoječi opis primera funkcije ne opisuje dovolj natančno. Še vedno je potrebno opisati tiste funkcionalnosti, ki jih opisi primerov ne zajamejo ter nefunkcionalne zahteve.

- **Primeri uporabe in funkcionalne zahteve:** Druga možnost je, da opišemo skope primere uporabe ter jih kasneje dopolnimo v funkcionalnih zahtevah. Tu je potrebno paziti, da posamezen primer uporabe povežemo s funkcionalnostjo sistema.
- **Le funkcionalne zahteve:** Naslednja možnost je, da ustvarimo le funkcionalne zahteve. Morebitne primere uporabe pretvorimo v funkcionalne zahteve ter jih primerno opišemo.

V primeru sistema Kurnik bodo funkcionalne zahteve temeljile na primerih uporabe, ki jih bomo dopolnili s poslovnimi pravili. V njih bo opisana večina posameznih funkcij, ki so potrebne za sistem.

3.1.6 Funkcionalne zahteve sistema Kurnik

Sistem naj bi implementiral naslednje funkcije:

Upravljanje s člani

Prioriteta: Najvišja
Primeri uporabe:

- Vpis člana v ŠK
- Podaljšanje statusa člana v ŠK
- Izpis člana iz ŠK
- Urejanje podatkov člana ŠK

Poslovna pravila: ŠK1, ŠK2, ŠK3, ŠK6, ŠK7
Poleg tega naj sistem vsebuje:

- **Pregled in filtriranje članov v tabeli:** V tabeli naj bodo vsi člani, trenutno vpisani v ŠK. Prikazani naj bodo ime, priimek, starost, naslov in trenutno stanje člana. Tabela naj bo filtrirna po naraščajoči ali padajoči vrednosti vseh stolpcev.

- **Iskanje člana preko iskalnega okna:** Sistem naj implementira iskanje člana po imenu ali priimku. V zadetkih poleg imena in priimka prikaže tudi naslov za lažjo identifikacijo člana v sistemu.
- **Pregled posameznega člana:** Sistem naj implementira pregled posameznega člana z njegovimi osebnimi podatki ter podatki o izobrazbi člana.

Upravljanje z dogodki

Prioriteta: Visoka
Primeri uporabe:

- Ustvarjanje novega dogodka
- Prijava člana na dogodek
- Odjava člana z dogodka
- Urejanje podatkov dogodka
- Izbris dogodka

Poslovna pravila: ŠK3, ŠK4, ŠK5
Sistem naj upošteva pravili ŠK3 ter ŠK5 ob prijavi člana na katerikoli dogodek. Poleg tega naj sistem upošteva tui pravilo ŠK4 ob prijavi člana na plačljiv dogodek.
Sistem naj vključuje tudi iskalni seznam vseh dogodkov. Vsak vnos v seznam naj bo en dogodek s podatki o nazivu dogodka, času ter kraju dogodka ter trenutnem številu prijavljenih.

Upravljanje z aktivisti

Prioriteta: srednja
Primeri uporabe:

- Ustvarjanje aktivista

- Urejanje aktivista
- Izbris aktivista

Sistem naj nove vnose aktivistov v sistem sprocesira ter pošlje e-sporočilo novemu aktivistu v manj kot minuti. Po 30 dneh naj se nepotrjeni profili aktivistov avtomatsko izbrišejo.

3.1.7 Nefunkcionalne zahteve

Po Wiegersu so nefunkcionalne zahteve tiste, ki se ne nanašajo na posamezno funkcionalnost ali obnašanje sistema. Sem spadajo zahteve, ki določajo kvalitetne atribute, specificirajo standarde ter kvalitete, ki jih mora sistem doseči. Te je moč združiti v dve glavni skupini: [16]:

- **Kvalitete izvajanja** kot so varnost, uporabnost ter so vidne ob delovanju sistema.
- **Kvalitete razvoja** kot so možnosti testiranja, razširjanja in nadgrajevanja. Te so močno povezane ali celo pogojene s specifičnimi tehnologijami, ki jih sistem uporablja.

Tipični primeri nefunkcionalnih zahtev so:

- Podatkovna varnost, podatkovna intergriteta ter odpornost sistema na napake
- Varnostno kopiranje podatkov
- Vodenje dnevnikov izvajanja ter dostopa
- Operativna cena sistema
- Obnovitev po katastrofi
- Skalabilnost ter razširljivost

- Odzivni časi ter dopustni časi neobratovanja

Te kvalitete lahko po Wigersu razvrstimo v naslednje skupine [16]:

- **Razpoložljivost:** Do katere mere je potrebno, da je sistem na razpolago.
- **Namestitvenost:** Kako težko je namestiti, odstraniti ter ponovno namestiti sistem.
- **Integriteta:** Do katere mere sistem ščiti pred napakami ter izgubo podatkov.
- **Interoperabilnost:** Kako enostavna je izmenjava datotek z drugimi sistemi.
- **Zmogljivost:** Kako hitro in predvidljivo se sistem odzove na uporabnikove ukaze.
- **Zanesljivost:** Kako dolgo naj sistem deluje pred napako.
- **Robustnost:** Kako dobro naj se sistem odzove na nepredvidljive vnose.
- **Varovanje uporabnikov:** Kako dobro sistem varuje pred poškodbami uporabnika.
- **Varnost dostopa in podatkov:** Kako dobro je zavarovan dostop do sistema ter podatki v sistemu.
- **Uporabnost:** Kako lahko se je naučiti, pomniti ter uporabljati sistem.

Priporočljivo je, da se v zasnovi sistema prioritizira skupine nefunkcionalnih zahtev ter posamezne nefunkcionalne zahteve sistema. Poleg tega je ključno, da so zahteve izražene absolutno ter empirično. Zahteva kot je:

Sistem naj se na uporabnikovo zahtevo po prijavi odzove v doglednem času.
ni uporabna, saj ni natančno določena časovna enota. Nenatančno opisane

zahteve otežujejo načrtovanje, razvoj in testiranje sistema. Zahteva:

Sistem naj se na uporabnikovo zahtevo po prijavi odzove v časovnem intervalu 1 sekude.

natanko opisuje časovno enoto. Ta se lahko primerno upošteva pri načrtovanju, razvoju ter se kasneje primerno testira.
Nefunkcionalne zahteve so hkrati lahko tudi glavni faktor pri ceni sistema, času razvoja sistema ter kvaliteti sistema. Na videz nedolžna zahteva po visoki dostopnosti sistema lahko pomeni podvajanje infrastrukture ter drugačno sistemsko arhitekturo. Nekatere zahteve so lahko tudi nedosegljive, zato je pomembno, da se pred implementacijo sistema načrtovalec posvetuje z programerji, arhitekti ter drugimi deležniki, ki lahko potrdijo tehnično izvedljivost zahtev.

3.1.8 Nefunkcionalne zahteve sistema Kurnik

Raspoložljivost

Sistem naj bo razpoložljiv vse dni v tednu, celo leto. Izjemoma je sistem lahko nedosegljiv ob prednapovedanih servisnih posegih, ki ne smejo trajati več kot 2 uri. Odprava nenapovedane napake se mora zgoditi v 1 uri po prvi najavi napake.

Namestitvenost

- Sistem naj bo moč namestiti na virtualni strežnik s prednameščenim operacijskim sistemom Centos v manj kot uri. V ta čas spadajo namestitev varnostnih posodobitev, namestitev podatkovne baze MySql, namestitev požarnega zidu, namestitev spletnega okolja Node.js, spletnega strežnika Express ter zagon spletne aplikacije.
- Sistem naj bo moč prenesti na drug sistem v manj kot dveh urah. To vključuje varnostno kopiranje MySQL podatkovne baze ter ponovna

namestitev sistema drugje.

- Odstranitev sistema naj bo mogoča v manj kot 30 minutah.

Integriteta

- Vsi podatki, ki so potrebni za delovanje sistema, naj bodo shranjeni v podatkovni bazi MySQL. Baza naj se varnostno kopira vsak dan ob 23.00 uri. Varnostna kopija ne sme trajati dlje od 5 minut in mora biti shranjena na drugi fizični lokaciji.

- Po izvršeni varnostni kopiji naj se izvrši preverba kopije sistema na oddaljenem sistemu. Varnostna kopija se šteje za uspešno, če se izhod algoritma MD5 ujema na izvirnem in oddaljenem sistemu.

- Sistem naj nad vsakim vnosom podatkov vrši preverjanje vnosa ter ščiti uporabnika pred nesmiselnim ali škodljivim vnosom v sistem.

Interoperabilnost

- Zaledni del sistema naj z osprednim delom komunicira preko REST spletnega vmesnika. Ta naj omogoča izmenjevanje podatkov z različnimi sistemi in naj bo neodvisen od osprednega dela sistema.

Zmogljivost

- Sistem naj se na vsak uporabniški ukaz odzove v največ 200ms. To je hkrati tudi časovna meja za nalaganje novih spletnih strani, avtorizacijo pri prijavi ali drugih operacijah.

- Operacije, ki zaradi tehničnih omejitev trajajo dlje naj po 200ms uporabniku prikažejo sporočilo z indikatorjem napredka. Za vsako tako operacijo naj bo izdelano strokovno mnenje ter podani razlogi za daljše trajanje operacije.

Zanesljivost

- Sistem naj povzroči manj kot 5 mehkih napak v 1000 primerih uporabe funkcionalnosti sistema. To so napake, kjer sistem zazna napako ter o tem takoj obvesti uporabnika sistema, da lahko uporabnik sistema ponovi primer uporabe sistema.

- Sistem naj povzroči manj kot eno trdo napako na mesec ob običajni uporabi. To je napaka, ki povzroči ustavitev delovanja ter posredovanje administratorjev oz. skrbnikov sistema.

Robustnost

- Sistem naj po shrambi, spremembi, odstranitvi ali dodajanju dogodka, izobrazbe ali člana poda možnost razveljavitve operacije, ki naj bo na voljo 10s.

Varovanje uporabnikov

Sistem nima posebnih zahtev glede varovanja uporabnikov sistema pred poškodbami ali škodo uporabnikov.

Varovanje podatkov

- Sistem naj vrši redne varnostne kopije podatkovne baze, kakor je določeno v razdelku podatkovne intergritete.

- Sistem naj posodobi CentOS operacijski sistem ter vse nameščene programske pakete najmanj enkrat na mesec.

- Sistem naj izdeluje varnostne kopije operacijskega sistema strežnika najmanj enkrat na teden.

- Koda, ki poganja spletni sistem, naj bo gostovana na ločenem strežniku.

- Dostopi do spletnega strežnika, podatkovnega strežnika in strežnika s programsko kodo so dostopni le sistemskemu administratorju ter administratorju podatovne baze. Vsak dostop do sistema naj se beleži v dnevnikih dostopa. Vsaka oseba naj ob vstopu do sistema zabeleži tudi razlog za dostop.

- Dolžina gesla za spletni strežnik, podatkovni strežnik ter strežnik z izvirno kodo naj bo več kot 20 znakov z najmanj eno številko in posebnim znakom.

- Dolžina gesla za dostop do informacijsega sistema, ki ga uporabljajo aktivisti, naj bo najmanj 8 znakov z najmanj eno številko ali posebnim znakom.

Uporabnost

Vsaka osnovna operacija, ki je našteta v opisih primerov uporabe, naj bo izpolnjena v največ eni minuti na najmanj 90 odstotkih testne množice brez danih navodil, izključujoč navodila naloge. V ta čas se ne upošteva čas vnosa različnih podatkov.

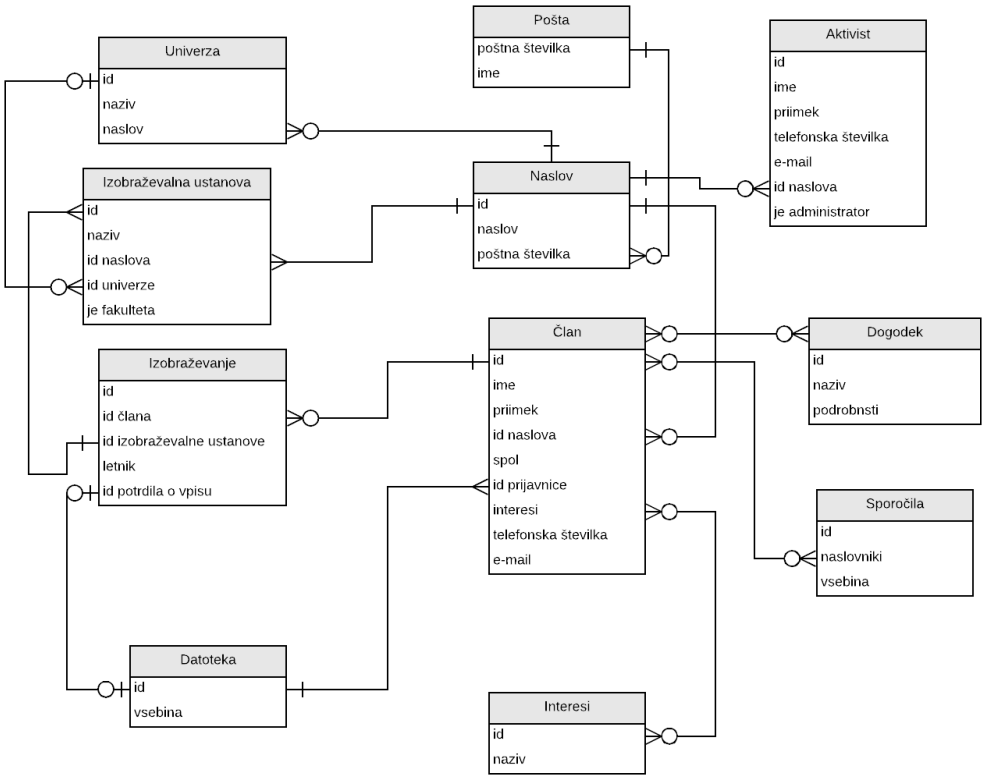
3.2 Podatkovni model

Informacijski sistemi uporabljajo različne podatke tekom svojega delovanja. Imena in priimki, EMŠO številke, različne identifikacijske številke, cene in količine, kraji in hišne številke so pogosti gosti informacijskih sistemov.
Ko izdelujemo nov informacijski sistem torej obdelujemo različne podatke. Da bi lahko te podatke varno, učinkovito in smiselno uporabljali, moramo vedeti, kako so podatki med seboj povezani, ter kako jih bomo predstavili v informacijskem sistemu. To nalogo opravljajo podatkovni modeli. Eden izmed njih - diagram entiteta-razmerje (angl. entity-relation diagram, krajše ERD) podatke združuje v logične skupine ter prikazuje, kako so podatki med seboj povezani za neko problemsko domeno.

Entitete predstavljajo fizični objekti (tudi osebe) ali agregacije podatkov, ki so pomembni za problem. Entitete so torej osebki, ki nastopajo v opisih funkcionalnosti in primerih uporabe. V diagramu so predstavljeni z naslovi kvadratov v diagramu. Med pretvorbo v relacijsko podatkovno bazo se entitete pretvorijo v eno ali več tabel. Število končnih tabel je odvisno od tega, kako so entitete med seboj povezane oz. odvisne.
Vsaka entiteta je opisana z enim ali več atributov. To so različni podatki, ki jih lahko vsebuje entiteta. Npr. vsaka pošta vsebuje naslov ter poštno številko. V podrobnejši obliki ER diagrama so predstavljeni tudi podatkovni tipi, ki pripadajo posameznim atributom.
Povezave med tabelami so razmerja. Razmerja označujejo povezavo entitet oz. njihovo medsebojno odvisnost. Tu so prav tako pripisane številnosti razmerij. Diagram 3.2 prikazuje enostaven ER diagram za študentski sistem Kurnik. To je podlaga za nadaljnjo analizo in pretvorbo v logični model podatkovne baze. Iz tega kasneje nastane definicija podatkovne baze, ki je uporabljena za ustvarjanje fizične podatkovne baze v podatkovnem sistemu MySQL.
Iz diagrama 3.2 je razvidno, da v sistemu nastopajo naslednje osnovne entitete:

- Člani
- Aktivisti
- Dogodki
- Izobraževanja
- Sporočila

To spovpada s primeri uporabe (diagram 3.1.4). Nad vsako osnovno entiteto se bodo vršile osnovne operacije branja, pisanja, ustvarjanja in brisanja (angl. create, read, update, delete, krajše CRUD). Ostale tabele služijo za povezavo ali stransko funkcionalnost.

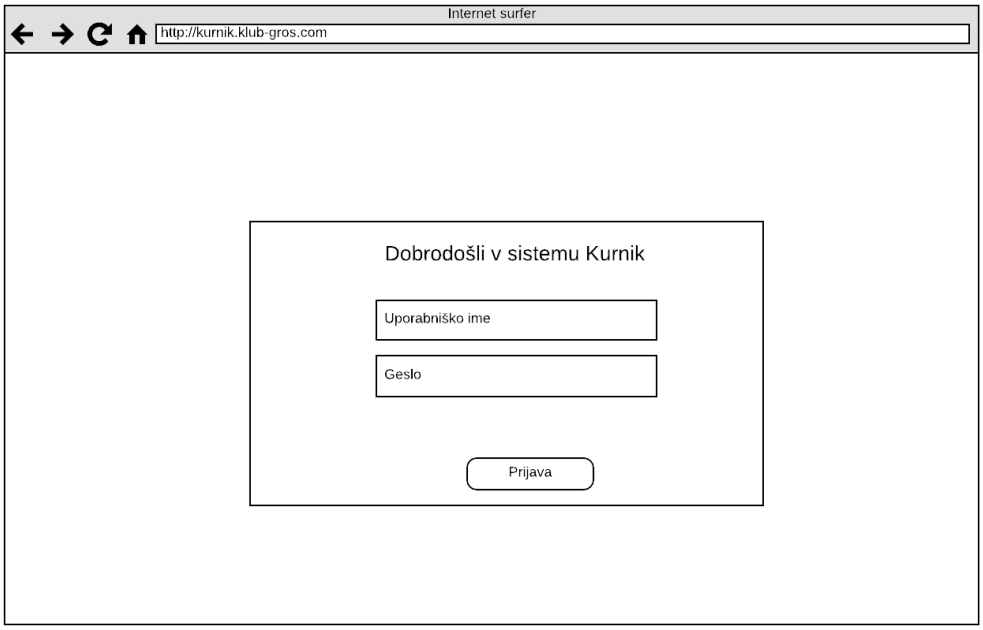


Slika 3.3: ER diagram

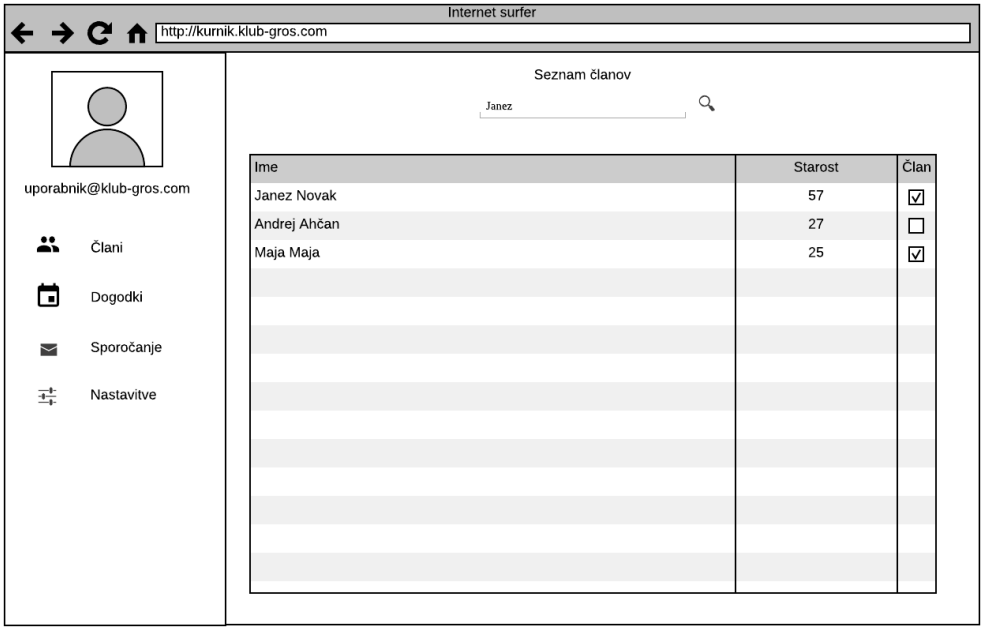
Diagram ER je pomemben, saj poleg enotnega modela podatkov, ki ga uporablja razvojna ekipa, pomaga odkriti morebitne pomanjkljivosti v funkcionalnih zahtevah. Če nekega osnovnega koncepta ni med funkcionalnimi zahtevami, je velika verjetnost, da je bila funkcionalnost zgrešena. Pri razvoju zahtev je torej ključnega pomena dobra ter stalna komunikacija med stranko ter razvojnikom zahtev.

3.3 Model uporabniškega vmesnika

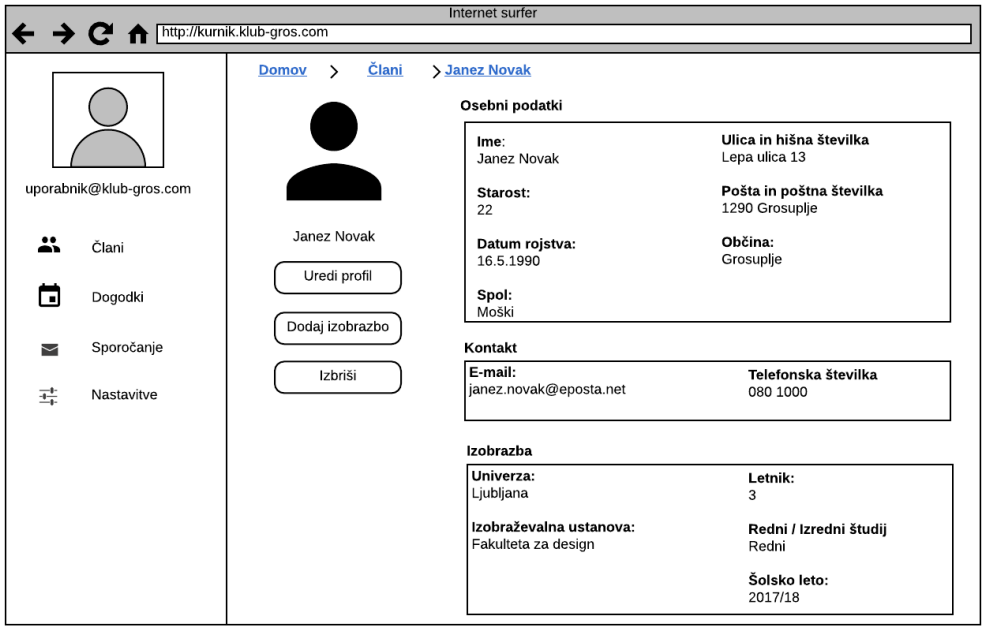
Dobro načrtovan model uporabniškega vmesnika prikazuje predviden izgled ter funkcionalnosti bodočega sistema. Model uporabniškega vmesnika je izredno dragocen, saj pomaga odkriti morebitne pomanjkljivosti in napake v načrtovanju sistema zgodaj v razvoju. Stranka si lahko s pomočjo modela predstavlja, kako bo ta izgledal ter skupaj z razvijalci opazi napake in pomanjkljivosti.
Zgodnje odkrivanje napak je izredno pomembno, saj je napake v zgodnji fazi mnogo lažje odpraviti kakor pozno v razvoju. Poleg tega pomaga potrjena skica uporabniškega vmesnika omejiti nevarnost zavrnitve sistema. Skica ilustrira pričakovanja razvijalcev in strank glede končne rešitve. Solina ocenjuje, da je čas, ki je potreben za popravek napake lahko tudi do 100-krat večji, če se napaka odkrije v produkciji namesto v načrtovanju [15].
Modeli uporabniškega vmesnika so lahko vse od enostavnih papirnatih skic elementov sistema do pol delujočih interaktivnih računalniških prototipov. Podrobnost izdelave modelov je odvisna od podrobnosti funkcionalnih zahtev, zahtev stranke, velikosti sistema ter medologije razvoja.
V diplomski nalogi bomo demonstrirali tehniko žičnega okvirja (angl. wireframe diagram). To je groba skica elementov sistema, njihove postavitve in njihovega okvirnega izgleda.



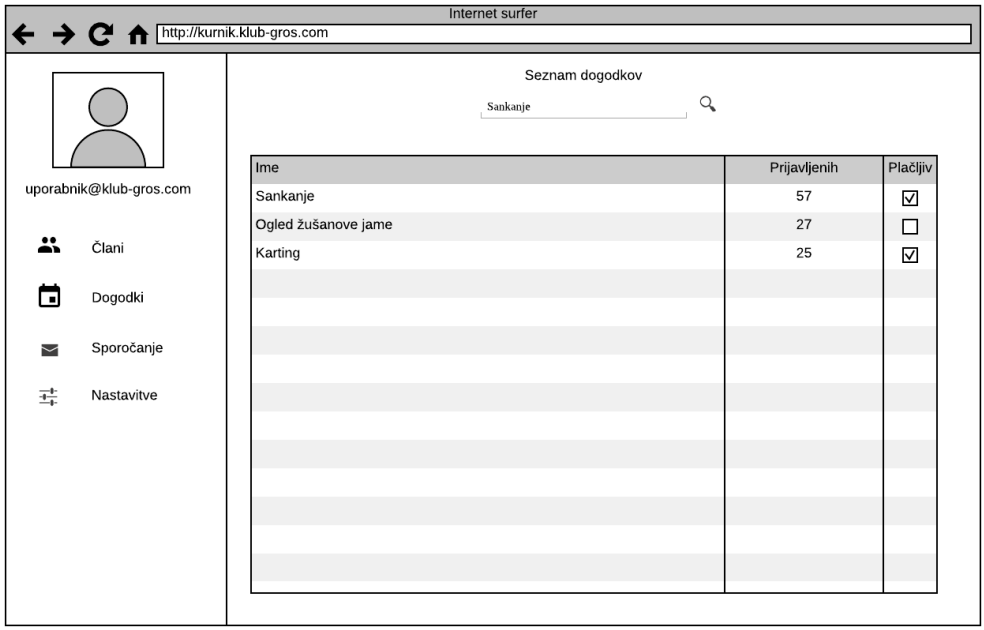
Slika 3.4: Žični diagram: prijava



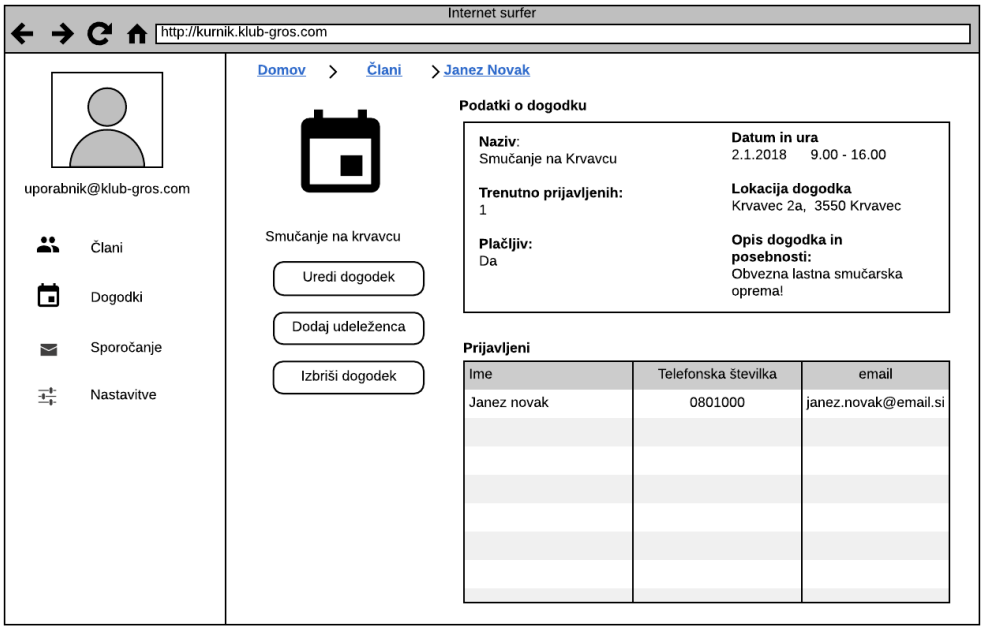
Slika 3.5: Žični diagram: seznam članov



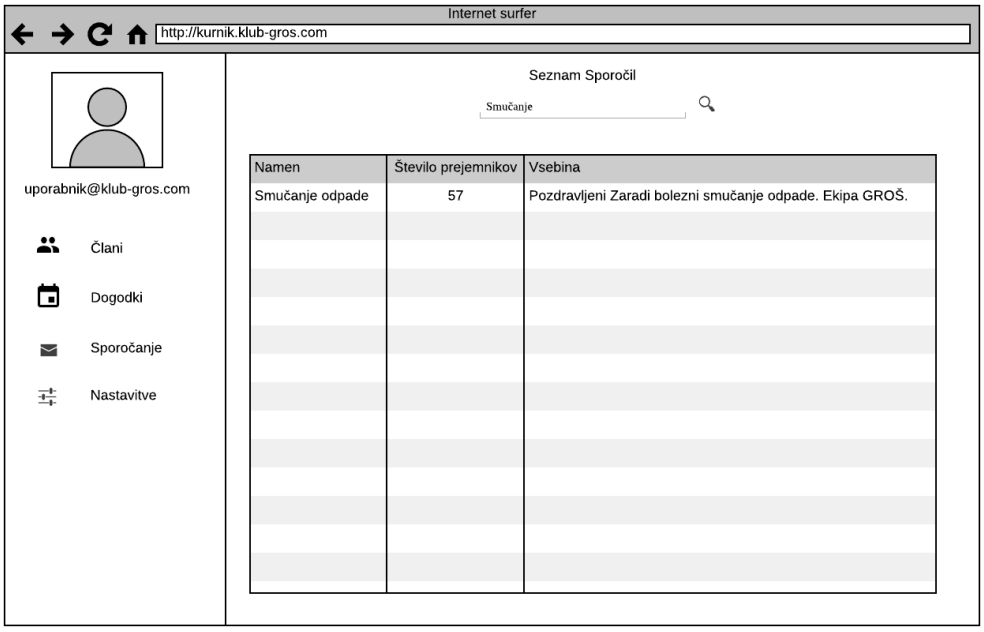
Slika 3.6: Žični diagram: podrobnosti člana



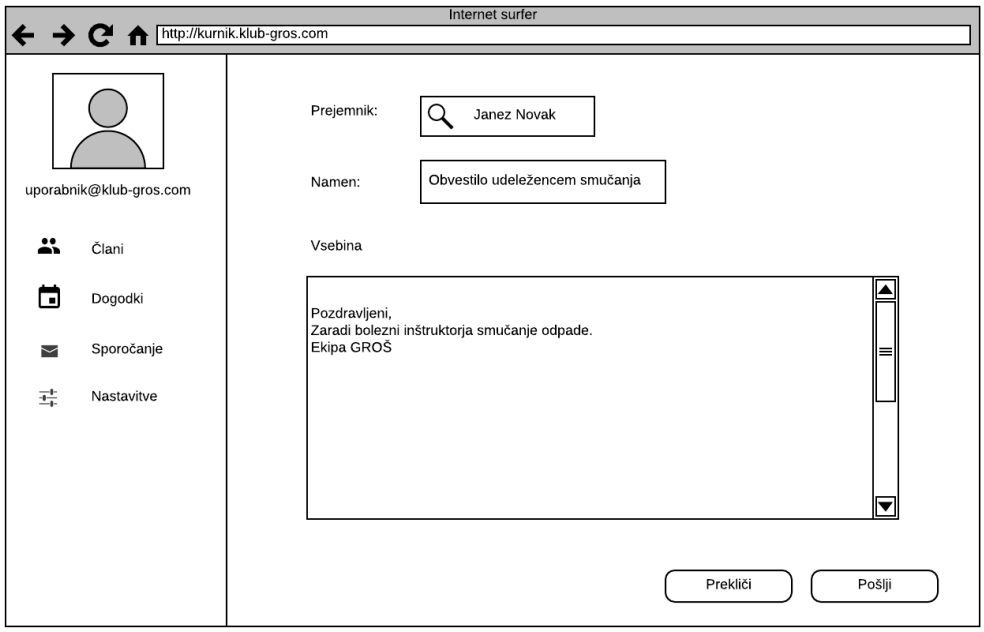
Slika 3.7: Žični diagram: seznam dogodkov



Slika 3.8: Žični diagram: podrobnosti dogodka



Slika 3.9: Žični diagram: seznam sporočil



Slika 3.10: Žični diagram: urejanje sporočila

Žični diagram uporabniškega vmesnika

Iz skic uporabniškega vmesnika je razvidna pozicija ter delno tudi namen in obnašanje elementov. S pomočjo teh skic si stranka mnogo lažje predstavlja izgled sistema ter ga potrdi oz. zavrne preden sistem preide v razvoj in testiranje.

Skice prikazujejo seznam članov s funkcijo iskanja, pregled posameznega člana, informacije o posameznem članu ter pregled dogodkov. Tu lahko vidimo, da se tudi na relativno enostavnih primerih informacijskih sistemov s pomočjo skic uporabniškega vmesnika hitro odkrijejo deli sistema, ki niso del funkcionalnih zahtev, vendar so del sistema kot logična posledica. V tem primeru je bil to navigacijski meni na desni strani ter sama organicazija strani. To je seveda moč organizirati tudi drugače, glede na izražene želje ter mnenja stranke ob predstavljeni skici.

Poglavje 4

Implementacija informacijskega sistema

Izdelani model informacijskega sistema je poskrbel, da je bodoč sistem dobro in nedvoumno opisan ter pripravljen na implementacijo. Vendar se tu pojavijo vprašanja glede samih tehnologij, sistemske arhitekture in razvojnega okolja, ki jih bomo uporabili za implementacijo. V tem poglavju bomo obdelali razloge za izbrane tehnologije, in predstavili razvojno okolje v katerem lahko učinkovito ustvarimo informacijski sistem.

4.1 Izbrane tehnologije in razvojno okolje

4.1.1 Sistemska arhitektura

Sistemska arhitektura je model, oz. predstavitev sistema, ki vključuje vse različne komponente sistema. Sistemske arhitekture so lahko abstraktne - definirajo različne sisteme, ki medsebojno delujejo v širšem sistemu ali pa so konkretne - določajo mesto in namen različne programske ter strojne opreme v sistemu.

Opis sistema Kurnik nam pove, da gre za spletno aplikacijo. Poleg tega je zaželeno, da je čim več komponent sistema neodvisnih med seboj, kar omogoča lažje morebitne spremembe sistema, enostavno vzdrževanje in prin-

cip ločevanja nalog (angl. seperation of concerns). Zaradi lažjega razvoja so bile zahtevane arhitekture, ki so dobro dokumentirane. Odločili smo se za trinivojsko arhitekturo spletnega sistema. To je najbolj razširjena arhitektura spletnih aplikacij. Deli se na prezentacijsko plast, ki prikazuje samo spletno aplikacijo ter sprejema ukaze, logično plast, ki skrbi za obdelavo zahtevkov prezentacijske plasti in podatkovno plast, ki skrbi za hrambo ter organizacijo podatkov [3].

4.1.2 Programska oprema

Programska oprema je bila izbrana glede na primernost, razširjenost uporabe, odprtokodnost ter dokumentiranost. Zato so vsi programski paketi, ki jih sistem Kurnik uporablja odprtokodni in široko uporabljeni. Različni programski paketi skrbijo za realizacijo različnih plasti. Prezentacijski sloj skrbi za predstavitev spletne aplikacije v tem primeru prevzame ogrodje AngularJS. Logični sloj skrbi za procesiranje zahtev, ki prihajajo z prezentacijskega sloja ureja spletni strežnik Express ter okolje NodeJS. Ta hrani podatke v t.i. podatkovni plasti, ki jo realizira podatkovna baza Oracle MySQL.

4.1.3 Strojna oprema

Strojna oprema, ki jo uporablja nek sistem, je večinoma odvisna od nefunkcionalnih zahtev, ki določajo kvalitete sistema. Če je zahteva sistema neprekinjeno delovanje, se taka zahteva odrazi v duplikaciji strojne opreme za primere nezgode in ločenega sistema za upravljanje s primarnim in sekundarnim sistemom.
Strojna oprema je prav tako lahko odvisna od funkcionalnih zahtev. (primer: v sistemu, ki je namenjen alkotestiranju, bo z veliko verjetnostjo potrebna posebna strojna oprema za izvajanje meritev.)
V primeru spletnega sistema Kurnik ne potrebujemo posebne strojne opreme.

Posamezne nivoje arhitekture predstavljajo različni programi (ali ločeni sistemi). Nujno potrebna strojna oprema je le en ločen fizični ali virtualni računalnik. Nanj lahko namestimo posamezne programske pakete, ki predstavljajo različne nivoje, lahko pa te programske pakete in z njimi nivoje delimo na ločene virtualne ali fizične strežnike odvisno od potrebe in kapacitet strežnikov [3].

4.2 Analiza diagrama primerov uporabe

Analiza primerov uporabe je ključni korak v razvoju modela informacijskega sistema, saj dotedaj dokaj neformalne primere uporabe razčleni ter pripravi za razvoj. Primere uporabe, ki niso jasni oz. so nepopolni, skupaj z razvojniki ali programerji dopolnimo ter odpravimo morebitne nejasnosti. Nenujni primeri uporabe, ki ne vplivajo na sam sistem, so iz analize izključeni [14].

4.2.1 Dopolnitev opisov primerov uporabe

Hiter pogled na diagram primerov uporabe ter opise primerov uporabe pokaže, da so nekateri primeri uporabe zelo enostavni oz. atomarni. Te v dopolnitvi izpustimo. Ti so:

- Preverba izobrazbe člana
- Tiskanje potrdila
- Plačilo dogodka
- Pošiljanje potrditvenega e-poštnega sporočila

Seznam potrebnih informacij (npr. pri ustvarjanju novega dogodka ter vpisu novega člana) se lahko ustvari s podrobnim branjem opisa problemske domene in z analizo modela podatkovne baze.

4.2.2 Identifikacija potrebnih razredov za realizacijo primerov uporabe

Objektno usmerjene tehnologije tipično uporabljajo razrede za realizacijo posameznih ključnih entitet. V primeru sistema Kurnik smo se odločili za tehnologije, ki niso tipično razredne. Zato bomo namesto osnovnih razredov govorili o osnovnih entitetah, ki jih je potrebno podpreti oz. zanje realizirati CRUD operacije.
Ob pregledu diagrama primerov uporabe (3.1.4) in opisov primerov uporabe lahko identificiramo naslednje potrebne osnovne entitete:

- Član
- Dogodek
- Aktivist oz. uporabnik
- Sporočilo
- Izobrazba
- Izobraževalna ustanova

4.3 Implementacija informacijskega sistema

4.3.1 Vpliv tehnologije na izhodiščni model sistema

Izdelava podrobnega modela je pomembna, saj nam pomaga problem razdelati, ga celovito opisati in obdelati pred izdelavo. Eden glavnih ciljev je enoumno razumevanje predvidenega delovanja sistema. Sistem je sedaj potrebno tudi implemetirati v tehnologijah, ki so bile izbrane.
Različne tehnologije, sistemska ogrodja, knjižnice in programski jeziki prinašajo različne prednosti in omejitve. Tako lahko izbrana tehnologija vpliva na končno implementacijo sistema.
Tekom analize sistema so bile že izbrane tehnologije, za katere verjamemo,

da bodo za implementacijo predvidenega sistema najbolj primerne.
Model ni statična stvar, ampak se razvija tudi tekom razvoja. Ko razvijamo sistem glede na model, je pričakovano, da zaradi specifik tehnologij pride do sprememb v delovanju in posledično v modelu sistema. Pričakovano je tudi, da so te spremembe omejene na tehnične podrobnosti in ne spreminjajo predvidene funkcionalnosti sistema, če to ni nujno potrebno.
Spletna aplikacija Kurnik je zasnovana v troslojni arhitekturi. To pomeni tri diskretne sisteme, ki skrbijo za predstavitev podatkov, obdelavo podatkov ter hrambo podatkov. Poleg tega je ena izmed nefunkcionalnih zahtev implementacija REST vmesnika na zalednem sistemu. Ti dve omejitvi ključno določata arhitekturo ter delovanje celotnega sistema.

4.3.2 Zaledni sistem

Zaledni sistem je neodvisen od ostalih delov sistema. Zasnovan je kakor REST vmesnik ter implementiran v Node.js okolju. Za streženje spletnih zahtevkov skrbi spletni strežnik Express. Programiran je v jeziku Javascript.

REST Vmesnik

REST (angl. Representational state transfer) je način komunikacije, ki narekuje delovanje spletnih storitev, ki se uporabljajo v sodobnih spletnih aplikacijah. REST je primarno namenjen hitremu razvoju spletnih aplikacij, neodvisnosti zalednega sistema in interoperabilnosti [12][6].
REST uporablja t.i. GET, POST, PUT ter DELETE metode HTTP (angl. Hyper Text Transfer Protocol) protokola ter standarden način naslavljanja posamezne entitete, da doseže operacije ustvarjanja, urejanja, branja ter brisanja podatkov v sistemu.
Glavne prednosti vmesnika REST so:

- **Zmogljivost:** REST vmesniki so sposobni prenesi veliko število poizvedb, saj so poizvedbe atomarne ter enostavne. Obdelava podatkov je minimalna, saj vmensik v večini primerov le vrača podatke, ki so shra-

njeni v podatkovni bazi. Zaradi narave vmesnika je mogoče poizvedbe tudi predpomniti ter tako še dodatno povečati zmogljivost vmesnika.

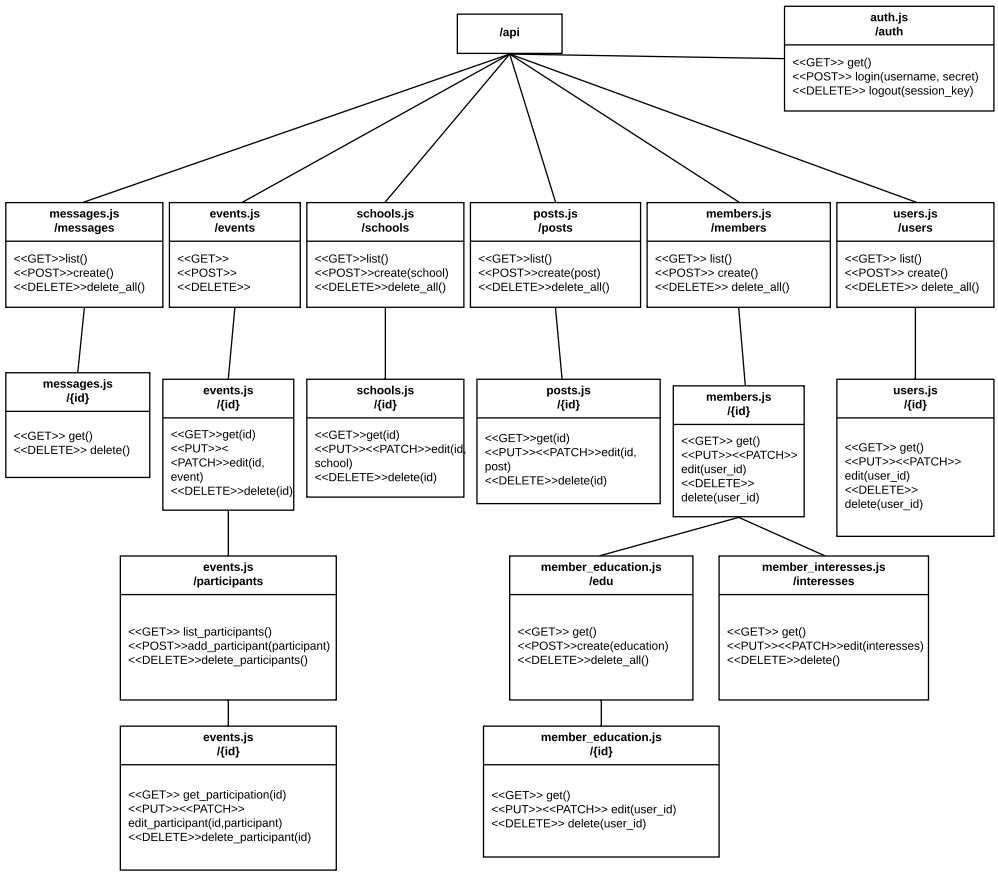
- **Preglednost komunikacije:** Komunikacija med spletnim strežnikom in odjemalcem je pregledna, saj so klici na zaledni sistem hierarhično strukturirani. V večini primerov podatke vračajo v berljivem JSON formatu.

- **Enostavnost vmesnika** Zahteve v vmesniku kličejo posamezne funkcije, ki so odgovorne za upravljanje z njimi. V veliki večini primerov gre za prodobivanje podatkov iz podatkovne baze ter vračanje le teh. To pomeni, da so funkcije enostavne in kratke. Poleg tega jih je mogoče z orodji kot so Postman enostavno testirati in razvijati ločeno od ospredenga sistema.

Uporaba REST sistema tako implementira podatkovno plast v tronivojskem sistemu. To pomeni, da je logična plast vmesnika na strani odjemalca, skupaj s predstavitveno plastjo. Vmesnik je neodvisen ter sporoben streči več odjemalcem hkrati, tudi če so odjemalci različnega tipa oz. različnih aplikacij.

Diagram zalednega sistema

Specifikacija UML ne predpisuje diagrama, ki bi natančno ponazoril delovanje REST vmesnika. Lahko pa s pomočjo diagrama komponent izdelamo diagram, ki ponazarja delovanje vmesnika ter hirarhijo posameznih funkcij.



Slika 4.1: Komponentni diagram končnih točk API vmesnika

Iz diagrama so razvidne posamezne datoteke, funkcije, pripadajoče HTTP metode ter hierarhija funkcij, ki upravljajo s podatki. Pri večini entitet se ponovi vzorec krovne končne točke, ki skrbi za seznam vseh entitet in ustvarjanje nove instance entitete ter končne točke z dodano identifikacijsko številko posameznega primera entitete v podatkovi bazi. Ta skrbi za branje, urejanje in brisanje posamezne entitete.

4.3.3 Ospredni sistem

Ospredni sistem je v našem primeru realiziran z pomočjo tehnologije AnuglarJS. Ta implementira elemente MVC (angl. model view controller) arhitekture na strani odjemalca.
Prezentacijsko (angl. view) plast aplikacije predstavljajo HTML in CSS datoteke, ki določajo izgled aplikacije. AngularJS vsebuje posebne izraze, ki vežejo pogled aplikacije z logično plastjo.
Logična (angl. control) plast aplikacije je v delu odjemalca, v javascript datotekah, ki določajo obnašanje aplikacije. Posamezne funkcije vežemo na dogodke v aplikaciji. Te funkcije urejajo tokove dogodkov, pošiljajo poizvedbe na strežnik, preusmerjajo uporabnika ali ga po potrebi pozovejo.
Poleg tega AngularJS podpira tudi modelni del MVC arhitekture (angl. model). V njem določimo pričakovan podatkovni model vhodnih podatkov. Ta omogoča preverbo pravilnosti vnosa podatkov v realnem času, kar je za izpolnjevanje obrazcev izredno pomembno. Čeprav je preverba podatkov del zalednega sistema in logike REST vmesnika tu vloge podvajamo zaradi pridobitev v uporabniški izkušnji, ki jo podvajanje prinaša. Poleg tega je podvajanje minimalno.

Implementacija žičnega diagrama

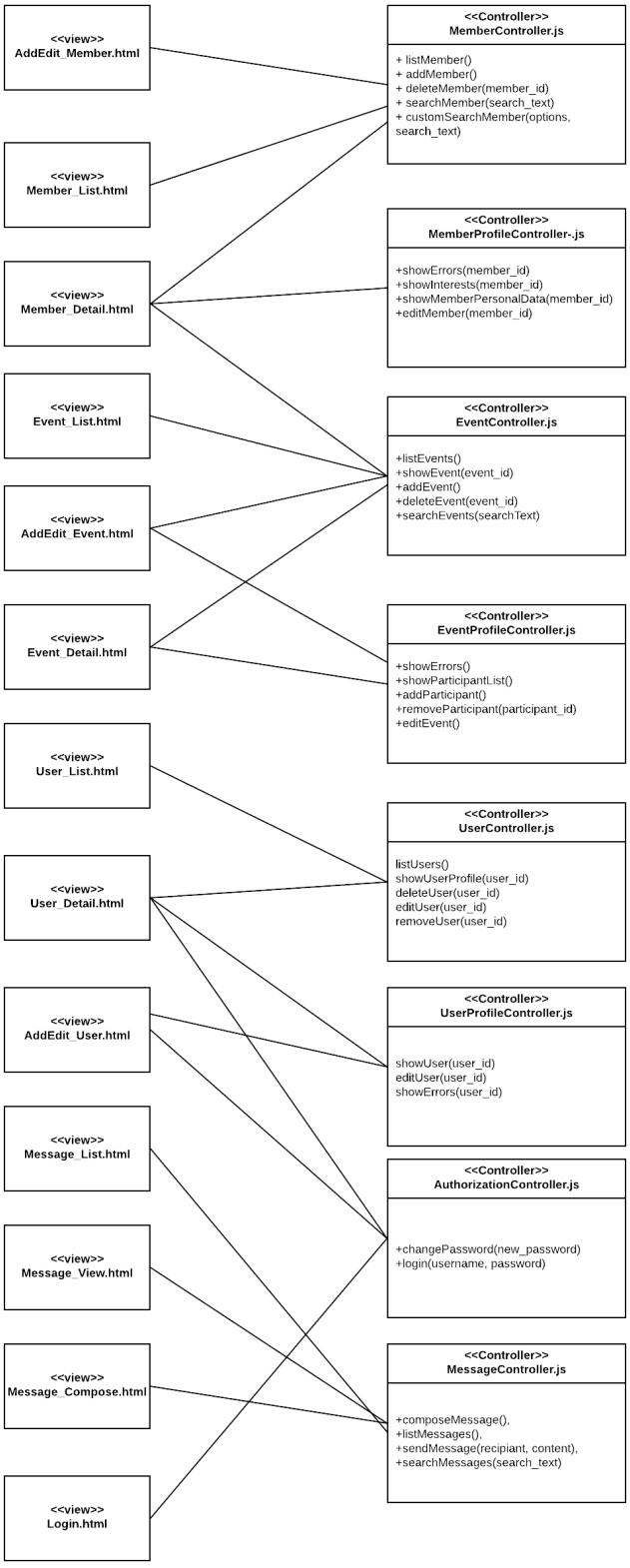
Žični diagram uporabniškega vmesnika narekuje 7 ločenih zaslonskih mask:

- Seznam članov
- Podrobnosti člana
- Seznam dogodkov
- Podrobnosti dogodka
- Seznam uporabnikov
- Podrobnosti uporabnika
- Prijava

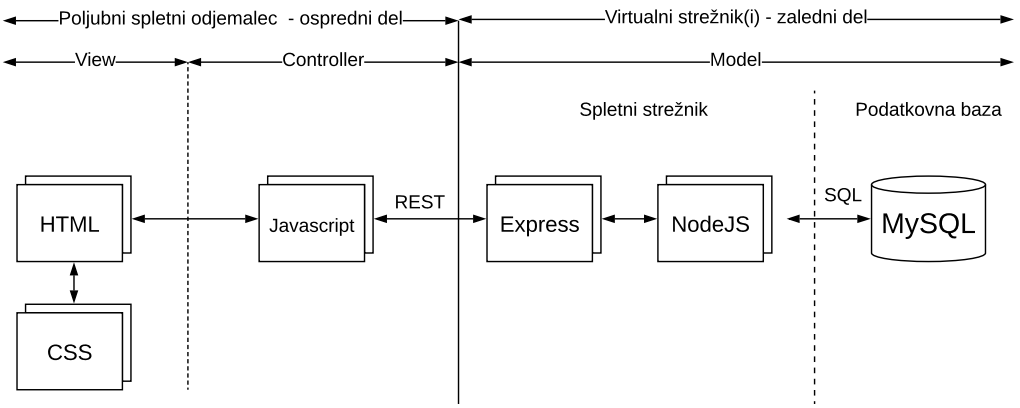
Poleg tega bodo zaradi specifike tehnologije AngularJS kot posebne zaslonske maske izdelane tudi zaslonske maske za dodajanje ali urejanje posameznega člana in dogodka. Vsaka posamezna zaslonska maska bo realizirana kot HTML datoteka, v kateri bo določen izgled sistema ter posamezni elementi. Ti elementi nato koristijo različne funkcije, ki so zapisane v javascript datotekah. Te realizirajo kontrolno plast aplikacije. Funkcije upravljajo z elementi na trenutni zaslonski maski, urejajo vnos informacij, izvajajo klice na REST vmensik in se nanje odzovejo.
Uporaba posameznega logičnega sklopa je razvidna iz naslednjega modela.

4.3.4 Podatkovna baza

Podatkovna baza je najenostavnejši del sistema, saj se komunikacija z bazo vrši po ustaljenem in razširjenem SQL (angl. structured query language) protokolu. Protokol je široko podprt, MySQL pa poleg hrambe podatkov omogoča tudi enostavno varnostno kopiranje ter po potrebi množenje podatkovne baze za dodatno varnost. V normalnem obratovanju sistema do podatkovne baze dostopa le zaledni sistem. Posamezna končna točka zalednega sistema se navezuje na osnovne entitetne tabele.



Slika 4.2: Komponentni diagram osprednega sistema



Slika 4.3: Komponentni diagram arhitekture sistema

Glede na ER model podatkovne baze smo ustvarili shemo podatkovne baze. Za to smo uporabili orodje MySQL Workbench, ki omogoča ustvarjanje shem, značilnih za SQL podatkovne baze. Tako nastalo shemo lahko izvozimo kot SQL datoteko, ki vsebuje navodila za izdelavo prazne SQL podatkovne baze.

4.3.5 Implementirana arhitektura sistema

Končna implementirana arhitektura sistema torej vključuje 2 diskretna dela - ospredni del, ki teče na spletnem odjemalcu in vključuje prezentacijsko ter logično plast in zaledni del, ki teče na virtualnem strežniku ter vključuje REST vmesnik ter tako implementira podatkovno plast.

Poraba sistemskih virov

V implementaciji informacijskega sistema se je po vzpostavitvi vseh potrebnih programskih paketov izkazalo, da je osnovni virtualni strežnik z dvema procesorskima jedroma hitrosti 2gHZ, 2 gigabajtoma pomnilnika ter 20 GB prostora na trdem disku več kot dovolj, saj sistem porablja od 5% do 15% procesorja in prav toliko pomnilnika. Poraba prostora je seveda odvisna v glavnem od velikosti posameznih programskih paketov in od velikosti podatkovne baze. Izkaže se tudi, da operacijski sistem CentOS z omenjenimi

programskimi paketi zaseda 3,5 GB prostora, podatkovna baza z vsemi podatki o slovenskih izobraževalnih ustanovah, poštah in s 1000 vpisanimi člani zaseda okrog 100 MB prostora na trdem disku.

Razširljivost

Arhitektura je zaradi modularne zgradbe in poudarka na osprednem sistemu učinkovita ter razširljiva.
Prva možna izboljšava zmogljivosti sistema je ločitev podatkovne baze in spletnega strežnika. To je možno narediti z vzpostavitvijo novega strežnika na ločenem sistemu ter preusmeriti zahtevke spletnega strežnika nov IP naslov ločenega strežnika in podatkovno bazo.
Naslednja možna izboljšava je uporaba aplikacij ali strežnikov za porazdelitev bremena (angl. load balancing). Z njimi lahko vzpostavimo več spletnih strežnikov in preko posebne programske ali strojne opreme za vsako sejo posebej usmerjamo promet proti spletnim strežnikom. Podobno možnost nudijo tudi plačljive različice podatkovne baze MySQL.
Bolj zapletena pa bi bila preureditev spletnega strežnika ter uporaba t.i. brezstrežniške tehnologije (npr. Amazon Lambda). Ta omogoča, da se programska koda poganja na strežnikih ponudnika tehnologije, ta pa skrbi za to, da je potrebna računska zmogljivost vedno na voljo in se dinamično prilagaja potrebam programa. [7].

4.4 Ugotovitve

Tekom naloge smo ustvarili dva glavna izdelka. Prvi je bil model informacijskega sistema. Temu je sledila izdelava informacijskega sistema glede na model.

4.4.1 Model informacijskega sistema

Izdelava modela informacijskega sistema je potekala predvidljivo. Opis problemske domene in modeliranje ni odvisno od določene tehnologije, kakor kasneje sama implementacija. Problemsko domeno najprej opišemo in jo modeliramo skladno z diagramskimi tehnikami standarda UML. Model po potrebi dopolnimo z drugimi diagramskimi tehnikami, katere z katerimi je razvojna ekipa seznanjena in pripomorejo k razumevanju problemske domene. Podrobnost diagramov in posledično modela je prepuščena lastni presoji. V predstavljenem primeru informacijskega sistema za podporo poslovanju študentskim klubom je število diagramov relativno majhno. UML specifikacija določa več kot 10 diagramskih tehnik za predstavitev različnih akspektov sistema.
V našem primeru smo uporabili le štiri. Razlog za omejeno podrobnost diagramov je omejenost problemske domene in enostavnost postopkov. V večini primerov gre za enostavne primere uporabe z manj kot 10 koraki. Podobno enostavna je procesna logika.
Vendar pa so diagrami - če tudi enostavni - izredno pomembni. Tudi v tem primeru so poskrbeli, da je problem celovito popisan. To zagotovi, da končen izdelek celovito podpre problemsko domeno.

4.4.2 Implementacija informacijskega sistema

Izdelava samega informacijskega sistema glede na model je bolj zapletena, saj je sistem odvisen od modela. Na posamezne tehnologije, ki smo jih uporabili so najbolj vplivale nefunkcionalne zahteve. Te določajo vrsto posamezne tehnologije in omejijo izbiro tehnologij, sistemski arhitektur in same sestave informacijskega sistema. Tudi v našem primeru so nefunkcionalne zahteve v veliki meri določale arhitekturo spletne aplikacije ter kje in v kakšni meri se nahajajo predstavitvena, logična ter podatkovna plast sistema.
Izbrane tehnologije prav tako niso tipično objektno usmerjene.
Ustaljen postopek pri objektno usmerjenem razvoju programske opreme s

pomočjo UML modelov je izdelava razrednih diagramov glede na primere uporabe. Identificirani razredi ter pripadajoče metode se nato prepišejo v programsko kodo, kjer se dopolnijo do delujoče kode in testirajo.

Neobjektna tehnologija z objektnim razvojnim pristopom

V našem primeru so uporabljene tehnologije, tipične za moderno spletno aplikacijo. Na žalost pa so neobjektne. To pomeni, da razredni diagram pride le delno v poštev. Analogi razredom so tako posamezne entitete v posamezni plasti. Ločitev ključnih entitet se najbolj izrazi v podatkovni in logični plasti sistema kot ločene datoteke, funkcije, dostopne točke programskega vmesnika ter tabele v podatkovni bazi.

Ločitev odgovornosti

Ločitev odgovornosti posamezne funkcije v posamezni plasti sistema se prav tako izvede različno od objektno usmerjenih tehnologij. Še vedno nadzorovan postopek se ne ukvarja z posamezno funkcijo v programski kodi temveč s posamezno funkcionalnostjo sistema. Ta se v večini primerov prevede v posamezno CRUD funkcionalnost, saj je večina funkcionalnosti sistema povezanih s hrambo, predstavitvijo ter vnosom informacij v sistem.

Ponavljajoči vzorci

V procesu izdelave takega sistema so identifikacija ključnih entitet, izdelava podatkovne baze, izdelava zalednega REST vmesnika in izdelava logične plasti (v našem primeru controller datotek ter funkcij) vzorci, ki se ponavljajo za vsako ključno entiteto. Ti so so si med seboj izredno podobni. Vse, kar se razlikuje, je podatkovni model posamezne entitete. To pomeni, da se enaki postopki identifikacije ključne entitete in podpiranja upravljanja s to entiteto v zalednem in logičnem delu osprednega sistema ponavljajo, ne samo skozi to aplikacijo, temveč skozi vse aplikacije, ki so zgrajene na podobni arhitekturi.

Druge sistemske arhitekture

Arhitektura našega sistema je izredno podobna t.i. MEAN arhitekturi. Edina razlika je v podatkovni plasti, kjer MEAN arhitehtura predvideva nerelacijsko MongoDB namesto relacijske podatkovne baze. Identifikacija, logična plast osprednega dela, zasnova ter velik del programske kode zalednega REST vmesnika bi ostali enaki, če bi Kurnik prevedli na MEAN arhitekturo.
Tudi v podatkovni plasti najdemo podobnosti med MEAN ter implementirano arhitekturo, saj v obeh primerih REST vmesnik podakte vrača v JSON formatu. Edina sprememba je v naslavljanju podatkovne baze.
To pomeni, da je mogoče podobnem postopek prevesti tudi na eno izmed najbolj razširjenih sistemskih arhutektur za razvoj spletnih aplikacij na svetu.

Poglavje 5

Zaključek

V diplomski nalogi smo opisali prednosti modeliranja v postopku izdelave informacijskega sistema. Prikazali smo, kako poteka zajem zahtev ter izdelava modela informacijskega sistema na problemski domeni študentskih klubov. Delo je potekalo skladno s metologijo RUP, modeli pa so v večini skladni s standardom UML.

Med izdelavo modela smo pridobili podroben vpogled v problemsko domeno ter problem podrobno popisali. To nam je omogočilo ustvariti sistem, ki je funkcionalno celovit. Proces izdelave modela je relativno enostaven, saj je problem abstrakten in ni odvisen od posameznih tehnologij. Model postane ključen del specifikacije, komunikacije med naročnikom in izvajalcem ter nepogrešljivo razvojno orodje. Podrobnost modela in njegova vsebina je odvisna od posamezne problemske domene, metologije dela in namena rabe sistema.

Implementacija modela je ponudila nekaj izzivov pri prevedbi objektno usmerjenega modela na neobjektno usmerjeno tehnologijo, vendar je moč objekte preslikati v ključne entitete, ter razviti velik del sistema glede na posamezne entitete te pa kasneje povezati v potrebne funkcionalnosti. Podoben postopek razvoja je mogoče uporabiti tudi pri drugih primerih razvoja informacijskega sistema s pomočjo modelov. Postopek ostane podoben tudi pri spremembi arhitekture.

Vse to nam omogoča, da ohranimo prednosti modeliranja informacijskih sistemov in hkrati uporabimo tehnologije, ki so trenutno v široki rabi in zadostujejo različnim potrebam uporabnikov glede na nefunkcionalne zahteve. Uporaba modelov nas ne omejuje na določene tehnologije ali sistemske arhitekture.

Uporaba sistema Kurnik in bodoče izboljšave

Po izdelavi informacijskega sistema Kurnik je ta že izšel in se uporablja za potrebe podpore poslovanju študentskih klubov širom Slovenije. Prve povratne informacije so pozitivne, saj sistem nadomešča veliko opravil, ki so bila ročna in zamudna.
Sedaj so na vrsti izboljšave sistema. Te lahko razvrstimo v tehnične izboljšave in funkcionalne izboljšave ter dodatke. Velika večina tehničnih izboljšav izhaja iz potreb in priložnosti, ki se odkrijejo ob uporabi sistema. Naslednji koraki vključujejo analizo dejanskih potreb (t.i. wants needs analiza) in analizo sedanje uporabe sistema. Dodatne funkcionalnosti se lahko odkrije tudi z dodatnimi metodami zbiranja podatkov kot so fokusne skupine, delavnice in zbiranje telemetričnih podatkov o uporabi osprednega in zalednega sistema[10].

Literatura

[1] Angular material. Dosegljivo: https://material.angularjs.org/latest/, 2018. [Dostopno 1.5.2018].

[2] Express. Dosegljivo: https://expressjs.com/, 2018. [Dostopno 1.5.2018].

[3] Multitier architecture. Dosegljivo: https://en.wikipedia.org/wiki/Multitier_architecture, 2018. [Dostopno 1.5.2018].

[4] Mysql. Dosegljivo: https://www.mysql.com/, 2018. [Dostopno 1.10.2018].

[5] Nodejs. Dosegljivo: https://en.wikipedia.org/wiki/Node.js, 2018. [Dostopno 1.5.2018].

[6] Representational state transfer. Dosegljivo: https://en.wikipedia.org/wiki/Multitier_architecture, 2018. [Dostopno 1.5.2018].

[7] Serverless computing. Dosegljivo: https://en.wikipedia.org/wiki/Serverless_computing, 2018. [Dostopno 1.10.2018].

[8] Uml. Dosegljivo: http://www.uml.org/, 2018. [Dostopno 1.5.2018].

[9] Wikipedia: Angularjs. Dosegljivo: https://en.wikipedia.org/wiki/AngularJS, 2018. [Dostopno 1.5.2018].

[10] Kathy Baxter and Catherine Courage. *Understanding Your Users: A Practical Guide to User Requirements Methods, Tools, and Techniques (Interactive Technologies)*. Morgan Kaufmann, 2005.

[11] Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web modeling language (webml): a modeling language for designing web sites. *Computer Networks*, 33(1-6):137–157, 2000.

[12] Xinyang Feng, Jianjing Shen, and Ying Fan. Rest: An alternative to rpc for web services architecture. In *Future Information Networks, 2009. ICFIN 2009. First International Conference on*, pages 7–10. IEEE, 2009.

[13] Martin Fowler. *UML Distilled*. Addison Wesley, third edition, 2005.

[14] Per Kroll and Philippe Kruchten. *The rational unified process made easy: a practitioner's guide to the RUP*. Addison-Wesley Professional, 2003.

[15] Franc Solina. *Projektno vodenje razvoja programske opreme*. Založba FE in FRI, 1997.

[16] Karl Wiegers and Joy Beatty. *Software requirements*. Pearson Education, 2013.